New Approaches to Classic Graph-Embedding Problems

Orthogonal Drawings & Constrained Planarity

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Thomas Bläsius

aus Konstanz

Tag der mündlichen Prüfung:	15. Juli 2015
Erster Gutachter:	Prof. Dr. Dorothea Wagner
Zweiter Gutachter:	Prof. Dr. Alexander Wolff

DOI: 10.5445/IR/1000048879

Danksagung

Meinen ersten nennenswerten Kontakt mit der theoretischen Informatik hatte ich in der Vorlesung mit dem nichtssagenden Titel „Informatik III“, deren heutiges Pendant den treffenden Namen „theoretische Grundlagen der Informatik“ trägt. Diese hervorragende, von Dorothea Wagner gehaltene Vorlesung hat nicht zuletzt dazu beigetragen, dass ich im späteren Verlauf meines Studiums möglichst viele von ihrem Lehrstuhl angebotene Veranstaltungen besucht und somit Algorithmik als Vertiefungsfach gewählt habe. Ihr Angebot im Anschluss an mein Studium zu promovieren war nicht nur Voraussetzung für diese Dissertation, sondern hat mir auch vier sehr lehr- und ereignisreiche Jahre in Karlsruhe beschert. Dabei konnte ich stets ohne zeitlichen Druck an den Probleme forschen, die mich interessierten, was nicht selbstverständlich ist. Dafür möchte ich mich ganz herzlich bei Dorothea bedanken.

Meinen Zweitgutachter Sascha (Alexander) Wolff traf ich das erste Mal bei der GD 2010 in Konstanz (auch wenn er sich daran vermutlich nicht mehr erinnern kann), bei der ich die Ergebnisse meiner Studienarbeit vorstellen durfte. Sein Kommentar zu meinem Vortrag war, man würde merken, dass ich mit großer Begeisterung dabei wäre, er hätte aber auf Grund der Geschwindigkeit nicht alles verstanden (ich wollte einfach zu viel in zu kurzer Zeit erzählen). Bei Sascha möchte ich mich für die Übernahme des Koreferats und insbesondere für die unkomplizierte Terminfindung zum Zwecke der Promotionsprüfung (wegen der er sehr früh aufstehen musste), sowie für seine hilfreichen Anmerkungen zu meiner Dissertation bedanken.

Großer Dank geht an Ignaz Rutter, der sowohl meine Studienarbeit (zusammen mit Marcus Krug) als auch meine Diplomarbeit betreut und mich während meiner Promotion fachlich angeleitet hat. Vielen Dank für die unzähligen nützlichen Anmerkungen zu meinen Aufschrieben und die sehr interessanten sowie spaßigen Whiteboard-Forschungs-Sessions. Ohne Dich wäre diese Dissertation nicht das was sie ist und ich hätte in den letzten vier Jahren deutlich weniger gelernt als ich gelernt habe.

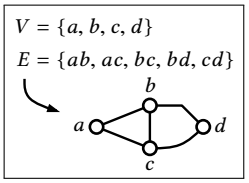
Moreover, I want to thank all my coauthors (listed below); it has been a great pleasure to work with you and I hope you also enjoyed working with me. Jawaherul Alam, Patrizio Angelini, Moritz Baum, Therese Biedl, Guido Brückner, Andreas Gemsa, Annette Karrer, Fabian Klute, Stephen Kobourov, Marcus Krug, Sebastian Lehmann, Benjamin Niedermann, Martin Nöllenburg, Roman Prutkin, Marcel Radermacher, Ignaz Rutter, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Alexander Wolff. Besides Ignaz Rutter, my sister Anne and my father Karl Hans, I also want to thank the many anonymous reviewers of my papers for proofreading parts of my thesis.

Bei meinen (aktuellen und ehemaligen) Kollegen Moritz Baum, Julian Dibbelt, Fabian

Fuchs, Andreas Gemsa, Michael Hamann, Tanja Hartmann, Andrea Kappes, Marcus Krug, Tamara Mchedlidze, Benjamin Niedermann, Martin Nöllenburg, Thomas Pajor, Roman Prutkin, Ignaz Rutter, Ben Strasser, Markus Völker, Franziska Wegner und Tobias Zündorf möchte ich mich für das angenehme Arbeitsumfeld bedanken. Ich habe nicht nur die Arbeit mit Euch genossen, sondern natürlich auch die Kickerrunden, die Fußballabende (sowohl vor dem Fernseher als auch im Soccer Center) sowie andere sportliche Aktivitäten wie Volleyball, Squash oder Badminton, die Schwimmbad- bzw. Baggersee-Ausflüge, Spieleabende, Kinobesuche, etc. Ohne Euch wäre die Zeit hier in Karlsruhe weit weniger schön gewesen.

Schlussendlich möchte ich meinen Eltern für ihre ständige Unterstützung danken. Euch habe ich das Privileg zu verdanken, ein so sorgenfreies Leben führen zu können.

Deutsche Zusammenfassung



In einer Welt, in der Daten im Übermaß verfügbar sind, ist es wichtig, über geeignete Verfahren zu verfügen, um vorhandenen Rohdaten analysieren, interpretieren und das so gewonnene Wissen kommunizieren zu können. Dabei erweisen sich Informationsvisualisierungen im Allgemeinen und Zeichnungen von Graphen (auch Einbettungen genannt) im Speziellen als hilfreich. Aus algorithmischer Sicht hat das Einbetten von Graphen zunächst im Zusammenhang mit dem Entwurf von integrierten Schaltkreisen Beachtung erlangt [AGR70]. Mittlerweile dient es meist dem Zweck der Informationsvisualisierung. Trotz der unterschiedlichen Anwendungen sind sich die Optimierungsziele oftmals erstaunlich ähnlich. Beispielsweise verschlechtern Kantenkreuzungen die Lesbarkeit von Zeichnungen erheblich. In Schaltkreisen führen sie dazu, dass die entsprechenden Leitungen auf unterschiedlichen Lagen verlegt werden müssen. Damit ist das Konzept der Planarität (also der kreuzungsfreien Einbettbarkeit) aus der Sicht verschiedener Anwendungen relevant.

Einige Einbettungsprobleme sind trotz ihrer langen Historie und ihrer grundlegenden Bedeutung bislang ungelöst. Das Ziel dieser Arbeit ist es, die Forschung an solchen klassischen Einbettungsproblemen voranzutreiben. Dabei steht die Entwicklung effizienter Algorithmen (mit polynomieller Laufzeit) im Vordergrund. Stellt sich für ein Problem heraus, dass es NP-schwer ist, so stelle ich diesem negativen Resultat immer auch positive Ergebnisse gegenüber. Dazu gebe ich beispielsweise Algorithmen an, deren Laufzeit nur exponentiell bezüglich eines oder mehrerer Parameter ist.

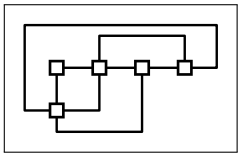
Die Arbeit gliedert sich in zwei Teile. Im ersten Teil werden sogenannte orthogonale Zeichnungen betrachtet. Dabei werden Kanten durch Sequenzen von ausschließlich horizontalen und vertikalen Strecken dargestellt. Im zweiten Teil geht es um die verallgemeinerten Planaritätsbegriffe c-Planarität (*engl.* clustered planarity) sowie simultane Planarität.

Orthogonale Zeichnungen

Historisch wurde die automatisierte Erstellung von orthogonalen Zeichnungen zunächst im Zusammenhang mit dem Entwurf von integrierten Schaltkreisen erforscht. Die naheliegendsten Optimierungskriterien waren dabei die benötigte Fläche, die sich direkt auf die Größe des Mikrochips überträgt, sowie die Gesamtkantenlänge. Etwas später kam die Knickzahl als mögliches Optimierungskriterium hinzu [Sto80].

Dies wurde zum einen motiviert durch Kosten, die an Knicken entstehen, wenn Informationen mittels Licht oder Mikrowellen übertragen werden, zum anderen durch „aufgeräumtere“ Zeichnungen. Der Aspekt der aufgeräumteren Zeichnung erlangt größere Bedeutung, wenn es darum geht, Graphen zum Zweck der Netzwerkanalyse anschaulich zu visualisieren. Dank der klaren und strukturierten Darstellung, die ausschließlich vertikale und horizontale Strecken mit sich bringen, gehören orthogonale Zeichnungen bis heute zu den meistverwendeten Zeichenstilen bei der Visualisierung von kleinen bis mittelgroßen Netzwerken.

Üblicherweise wird der Eingabegraph bei der Erzeugung orthogonaler Zeichnungen als planar vorausgesetzt. Für nicht-planare Graphen wird zunächst eine sogenannte Planarisierung mit möglichst wenigen Kreuzungen berechnet. Diese kann dann wie ein planarer Graph behandelt werden. Da man jeden Gitterpunkt im orthogonalen Gitter nur in vier verschiedene Richtungen verlassen kann, schränkt man sich bei orthogonalen Zeichnungen häufig auf Graphen mit Maximalgrad 4 ein. Eine Möglichkeit auch mit höhergradigen Knoten umzugehen bietet das sogenannte Kandinskymodell.

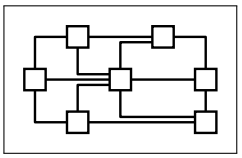


Graphen mit Maximalgrad 4. Trotz drei Jahrzehnten intensiver Forschung zu orthogonalen Zeichnungen blieben eine Reihe Fragen lange unbeantwortet. Beispielsweise ist seit 1994 bekannt, dass jeder 4-planare Graph (planar, mit Maximalgrad 4) eine orthogonale Zeichnung mit zwei Knicken pro Kante besitzt [BK98], es jedoch NP-schwer ist zu entscheiden, ob ein gegebener Graph ohne Knicke gezeichnet werden kann [GT01]. Erst 2010 konnte ein effizienter Algorithmus angegeben werden, der entscheidet, ob ein gegebener 4-planarer Graph eine orthogonale Zeichnung mit einem Knick pro Kante besitzt [Blä+14].

Um diese Komplexitätslücke weiter zu schließen, untersuche ich den Fall, dass manche Kanten nicht geknickt werden dürfen, wohingegen die übrigen Kanten jeweils mindestens einen Knick erlauben. Ich gebe einen parametrisierten Algorithmus an, dessen Laufzeit nur exponentiell in der Anzahl nicht zu knickender Kanten ist (die Laufzeit ist polynomiell, solange diese Anzahl in $O(\log n)$ liegt). Auf der anderen Seite zeige ich, dass das Problem NP-schwer wird, sobald der Graph $O(n^\epsilon)$ ($\epsilon > 0$) unknickbare Kanten enthält, selbst wenn diese gleichmäßig über den Graphen verteilt sind.

Bei den bislang erwähnten Einbettungsproblemen handelt es sich um Entscheidungsprobleme, bei denen nur überprüft wird, ob es eine Zeichnung mit den gewünschten Eigenschaften gibt. Existiert beispielsweise keine knickfreie Zeichnung, so würde man stattdessen gerne eine Zeichnung mit möglichst wenigen Knicken ausgeben (was NP-schwer ist, da es bereits schwer ist zu testen, ob es ohne Knicke geht). Durch den Beweis einiger struktureller Eigenschaften von orthogonalen Zeichnungen mit einem Knick pro Kante gelingt es mir, einen effizienten Algorithmus anzugeben, der die An-

zahl der Knicke minimiert, unter der Voraussetzung, dass der erste Knick jeder Kante keine Kosten verursacht. Dies ist auch dann noch möglich, wenn man jeder Kante eine individuelle konvexe Kostenfunktion zuweist (vorausgesetzt, der erste Knick ist kostenlos). Damit gebe ich den ersten effizienten Algorithmus zur Knickminimierung in dem vorliegenden Szenario an, der beliebige 4-planare Graphen als Eingabe erlaubt. Darüberhinaus ist dieser Algorithmus optimal in dem Sinne, dass das Weglassen einer der Forderungen (konvexe Kostenfunktionen und erster Knick pro Kante ist kostenlos) das Problem NP-schwer macht.

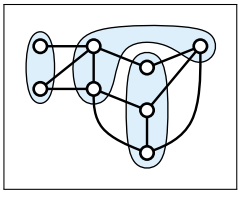


Kandinskyzeichnungen. Im Kandinskymodell werden Knoten als Quadrate fester Größe dargestellt und mehrere Kanten dürfen einen Knoten in dieselbe Richtung verlassen. Das Kandinskymodell wurde bereits 1995 vorgestellt [FK95]. Dabei wurde auch ein Algorithmus angegeben, der die Anzahl der Knicke minimiert, unter der Voraussetzung, dass die Topologie der Zeichnung bereits festgelegt ist (d.h. die zyklische Ordnung der Kanten um jeden Knoten ist gegeben). Wenig später stellte sich heraus, dass der Algorithmus einen Fehler enthält. Seither entstanden zahlreiche, meist approximative oder heuristische Verfahren, die Kandinskyzeichnungen generieren oder das Modell, zum Beispiel auf nicht-planare Graphen, erweitern.

Indem ich zeige, dass Knickminimierung im Kandinskymodell NP-schwer ist, beantworte ich die grundlegende Frage nach der Komplexität dieses Problems. Darüberhinaus betrachte ich seine parametrisierte Komplexität bezüglich verschiedener Parameter. Dies liefert insbesondere einen polynomiellen Algorithmus für serien-parallele Graphen (Laufzeit $O(n^3)$), sowie einen subexponentiellen Algorithmus im allgemeinen (Laufzeit $2^{O(\sqrt{n}\log n)}$). Damit beantworte ich nicht nur eine seit zwanzig Jahren offene Frage, sondern gebe auch neue algorithmische Lösungsansätze für dieses NP-schwere Problem.

Einbettung planarer Graphen unter Nebenbedingungen

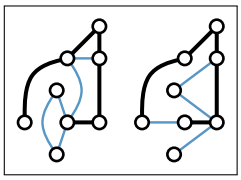
In diesem Teil meiner Arbeit geht es um zwei Probleme, bei denen es nicht genügt einen einzelnen gegebenen Graphen möglichst ansprechend und übersichtlich darzustellen.



C-Planarität. Ist zu dem Graphen noch eine Gruppierung (*engl.* clustering) der Knoten gegeben (beispielsweise die Einteilung von Klassen eines Softwareprojekts in Pakete), so kann es wünschenswert sein, neben dem Graphen selbst, auch diese Gruppierung mithilfe von Regionen darzustellen. Neben Kreuzungen zwischen Kanten, kann es in einer

solchen Zeichnung auch zu Kreuzungen zwischen unterschiedlichen Regionen oder Kreuzungen zwischen Kanten und Regionen kommen. Damit erhält man eine Verallgemeinerung des Konzeptes der Planarität: Ein gruppierter Graph heißt *c-planar* (*engl.* clustered planar), wenn eine Zeichnung existiert, in der keiner dieser drei Kreuzungstypen auftaucht. Das Problem, einen gegebenen gruppierten Graphen auf c-Planarität zu testen, wurde bereits 1986 das erste Mal betrachtet [Len89]. Trotz zahlreicher Arbeiten zu dem Thema konnten seither nur Spezialfälle gelöst werden. Die Komplexität des allgemeinen Problems ist nach wie vor ungeklärt.

Indem ich eine neue Datenstruktur zusammen mit einer Charakterisierung für c-Planarität angebe, kann ich zeigen, dass c-Planarität auf ein bedingtes Einbettungsproblem hinausläuft. Dabei stellt sich die Frage, ob ein gegebener planarer Graph eine planare Zeichnung besitzt, wenn man die möglichen Ordnungen von Kanten um Knoten herum einschränkt. Mithilfe dieser neuen Sichtweise zeige ich, dass sich diverse vorherige, auf den ersten Blick sehr unterschiedliche Resultate, mit ähnlichen Techniken beweisen lassen. Neben der Vereinheitlichung und Vereinfachung existierender Ergebnisse gebe ich effiziente Algorithmen für einige bislang offene Fälle an.



Simultane Planarität. Eine weitere Anwendung stellt die Visualisierung dynamischer Graphen dar. Hierbei möchte man neben der Graphstruktur auch die Veränderung dieser Struktur zwischen verschiedenen Zeitpunkten verstehen. Daraus ergibt sich das Problem der simultanen Visualisierung, bei der mehrere Graphen so gezeichnet werden sollen, dass ihr gemeinsamer, unveränderter Teil gleich dargestellt ist. Dabei soll weiterhin jeder der Graphen für sich eine möglichst übersichtliche Zeichnung haben. Wählt man die Kreuzungen zwischen Kanten als vorwiegendes Ästhetikkriterium, so erhält man das Konzept der simultanen Planarität. Dieses ist nah verwandt mit der c-Planarität, und es ist ebenfalls ein offenes Problem, ob man zwei gegebene Graphen effizient auf simultane Planarität testen kann, wohingegen diese Frage für diverse Sonderfälle positiv beantwortet werden konnte.

Bisher wurde meist angenommen, dass der gemeinsame Teilgraph zusammenhängend ist. Dies vereinfacht das Problem der simultanen Planarität insofern, dass es genügt, die beiden Graphen so einzubetten, dass die zyklischen Ordnungen der gemeinsamen Kanten um Knoten konsistent sind. Besteht der gemeinsame Graph aus mehreren Zusammenhangskomponenten, so muss man darüber hinaus konsistente relative Positionen der Komponenten zueinander sicherstellen. Ich gehe zunächst den umgekehrten Weg und löse die Fälle, in denen die zyklischen Ordnungen keine Rolle spielen, man also nur für konsistente relative Lagen sorgen muss. Die daraus entstehenden Techniken kombiniere ich mit bereits existierenden sowie neu von mir entwickelten Verfahren zur Sicherstellung konsistenter zyklischer Ordnungen.

Daraus ergibt sich insbesondere ein effizienter Algorithmus für den Fall, dass jede Zusammenhangskomponente zweifach zusammenhängend ist, Maximalgrad 3 hat oder außenplanar ist mit Schnittknoten von Grad höchstens 3. Ist jede Zusammenhangskomponente zweifach zusammenhängend, so hat der Algorithmus sogar optimale (lineare) Laufzeit.

Contents

1 Introduction

1.1 Motivation

In a world where information is available in abundance, being able to efficiently analyze and interpret the raw data at hand is crucial. In many cases, computers can take over that task. For example, a shortest-path algorithm can compute the fastest way to get from one place to another in a road network, and a minimum-cut algorithm can spot weak points in transportation, energy, or computer networks, facilitating reasonable decisions on where to invest into the infrastructure. However, algorithms processing given input data to compute and output optimal solutions have their limitations. In fact, such a direct algorithmic approach is not applicable if the question the user wants an answer for is to vague to be formalized in a problem statement, if a black-box solution is not acceptable (e.g., if the user not only needs an answer but also arguments for this answer), or if there exists no algorithm computing a solution in reasonable time.

The problem of finding the leader of a group of people based on their communication is an example for a question that is hard to formalize. Surely, there is a variety of centrality measures and choosing the most central person with respect to one of these measures can be a reasonable solution. However, different measures lead to different solutions and maybe none of the measures reflects the user's understanding of a person being the leader. In fact, the user himself may not know what his understanding of being a leader precisely is. Beyond the difficulties of formalizing this question, a black-box solution is probably not acceptable in most use cases. Similarly, a plan for an extension of the infrastructure that achieves maximum failure safety subject to a given budget may be of little value as political decisions play an important role in such projects. Moreover, many of these problems coming from real-world questions include fundamental NP-hard problems such as Steiner Tree and can thus not be solved in polynomial time assuming that $P \neq NP$.

Accepting that a direct algorithmic solution is not applicable does however not mean that a computer cannot help to answer a user's question. For example, architects make heavy use of computers although computers do not automatically generate complete blueprints. The key here is to integrate the user in the process of finding an answer to his question. In this way, a formal problem statement is no longer necessary and the user can change his mind on the exact goal during the process of finding an answer. As the user is integrated in this process, it is less likely that he questions the result than when it comes from a black-box algorithm. Finally, human intuition can

help to find the crucial steps leading to a solution for an NP-hard or even undecidable problem (e.g., automated theorem proving).

As humans are not good at working with large sets of abstract data, the raw data must be complemented by some visual representation. E.g., assume the user is presented a drawing of the communication graph of a group of people. With such a visualization, the user can get an overview over the data set, which enables him to find a group leader or argue for or against potential leaders suggested by an algorithm even without knowing what these algorithms actually compute or how they work. This leads to a general application of graph drawing, or more generally, information visualization: A user has a potentially unknown data set at hand and wants to accumulate knowledge based on this data set.

A slightly different application is the following. The user already has knowledge about a data set and wants to make this knowledge accessible to others. This is for example relevant for teachers, museums, and journalists who want to impart knowledge to students, visitors, and readers in a vivid manner. UML diagrams that visualize the structure of a software project also fall in this category.

Visualizing information and in particular drawing graphs to make knowledge accessible to others has a long tradition. In fact, the oldest known drawings of graphs date back at least 900 years [Lim14]. Figure 1.1 shows the drawing of a tree from 1866 visualizing a biological categorization.

It is not surprising that mathematicians who studied graph theory in general were also interested in drawings of graphs. To mention two classical examples, Kuratowski [Kur30] showed that a graph is *planar*, i.e., it can be drawn in the plane without crossing edges, if and only if it does not contain a subdivision of the K_5 or the $K_{3,3}$ as subgraph; see Figure 1.2a. Moreover, Wagner [Wag36] showed that every planar graph has a planar straight-line drawing (a result usually referred to as "Fáry's theorem" [Fár48]); also see Figure 1.2b. The proof of Fáry's theorem is actually constructive and could thus be used to generate planar straight-line drawings for planar graphs.

From an algorithmic point of view, the problem of drawing graphs first received attention not for the purpose of visualizing information but in the context of integrated circuits [AGR70]. Once the logical structure of a computer chip is designed, it has to be transferred into an actual physical structure, i.e., components and connections between them must be mapped to physical locations and wires between these locations, respectively. In other words, the graph representing the logical structure has to be drawn. A similar, more recent application is the design of biochips (also called lab-on-a-chip); see Figure 1.2c. A biochip is basically a small laboratory on which multiple biochemical reactions can be performed simultaneously.

In summary, besides visualizing information to make them accessible to humans, finding a physical representation for a given logical structure is another application for drawing graphs. In this context, one often finds *orthogonal drawings*, in which

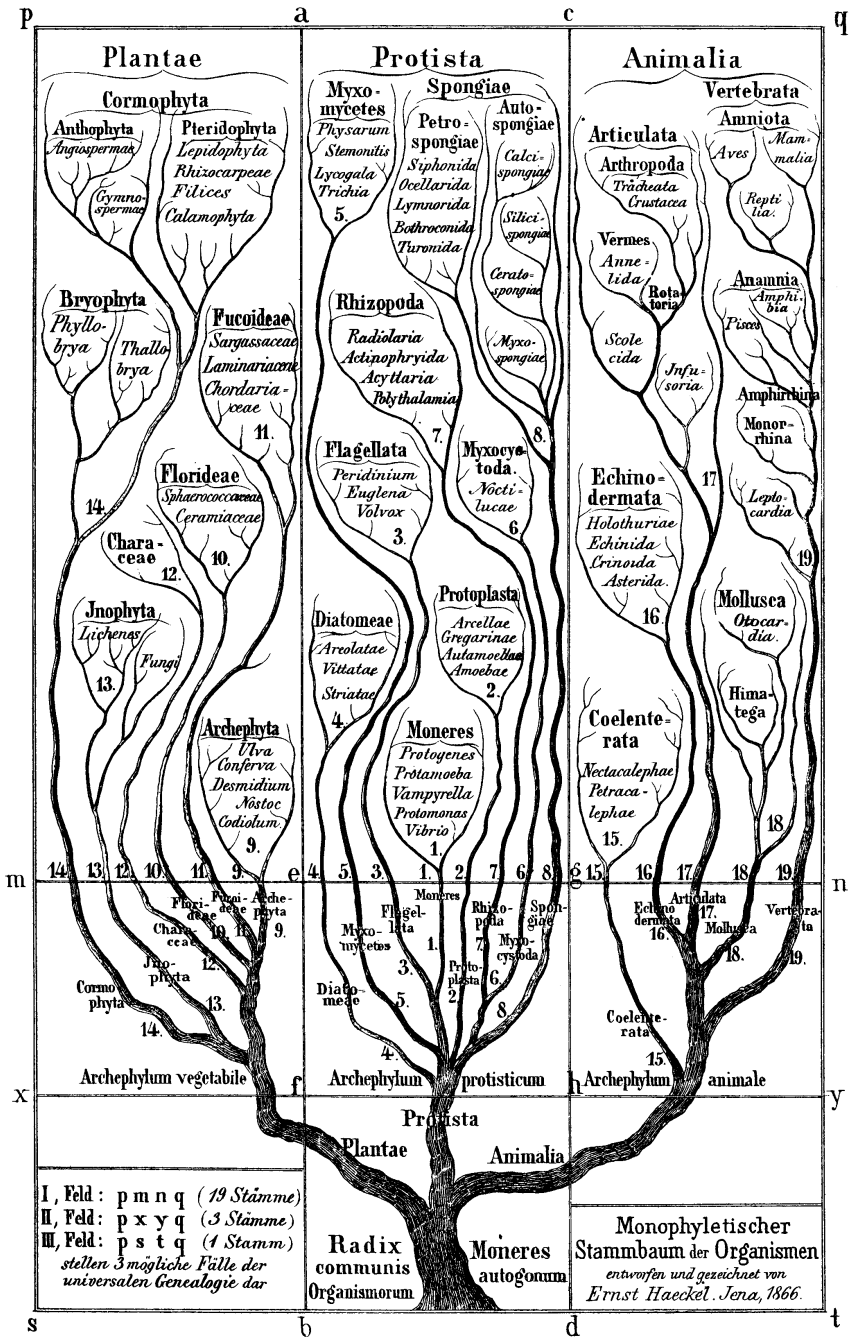


Figure 1.1: The "Monophyletic Family Tree of Organisms" by Ernst Haeckel from 1866. The three top-level categories (also called kingdoms) are plants (Plantae), unicellular organisms (Protista), and animals (Animalia). In the top right corner there are for example birds (Aves), reptiles (Reptilia), and mammals (Mammalia).

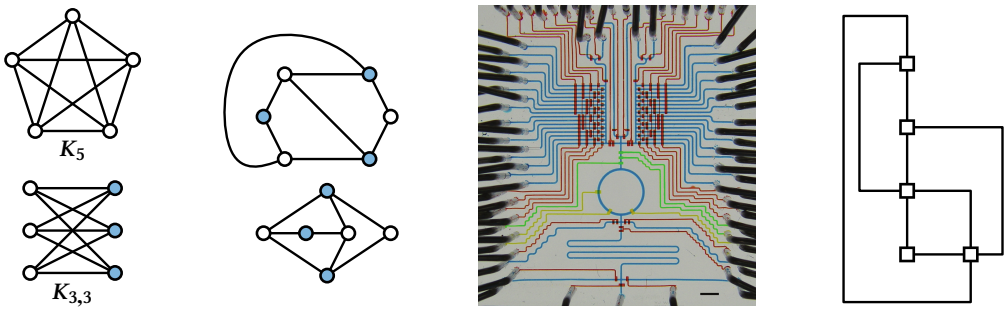


Figure 1.2: (a) The fundamental non-planar graphs K_5 (complete graph with five vertices) and $K_{3,3}$ (complete bipartite graphs with three vertices in each partition). (b) The $K_{3,3}$ becomes planar when removing a single edge (planar drawing in the top). By Fáry's theorem, the resulting graph has a planar straight-line drawing (bottom). (c) A biochip that can perform up to 1024 reactions in parallel [Wan+09]. (d) An orthogonal drawing.

the edges are represented by sequences of horizontal and vertical segments. E.g., in the biochip in Figure 1.2c, the edges are routed orthogonally (with rounded bends). Figure 1.2d shows a more abstract orthogonal drawing of a graph.

The optimization objectives that researchers first pursued when automatically generating orthogonal drawings were a small number of crossings and a small area [AGR70]. In the context of designing integrated circuits, these are natural optimization criteria as the area of the drawing directly translates into the area of the resulting chip and wires that cross need to be routed on different layers of the chip. Later, a third optimization criterion was introduced by Storer [Sto80]; the number of bends. Storer's motivation for minimizing the number of bends was threefold. First, minimizing the number of bends can serve as a heuristic for minimizing the area. Second, bends may cause costs, e.g., when information is transmitted via light or microwaves. And third, he noticed that drawings with fewer bends "appear simpler" and are thus possibly more desirable.

Due to the clear and structured appearance of axis-aligned segments, orthogonal graph drawing is one of the most popular drawing styles for network visualizations. Despite the fact that designing chips and visualizing networks are very different applications to graph drawing, the optimization criteria are actually very similar. As edge crossings heavily obfuscate a graph's drawing [PCJ96], minimizing the number of crossings is one of the most important optimization goals when visualizing graphs. Moreover, a small area makes sure that the drawing fits onto the output device (e.g., a computer screen or a piece of paper), and the above-mentioned aspect that drawings with fewer bends appear simpler becomes more important in the context of drawing graphs for the purpose of visualizing information.

However, a graph's drawing alone is sometimes not satisfactory even if it is aesthetically appealing and does a good job in revealing the graph's structure. For example

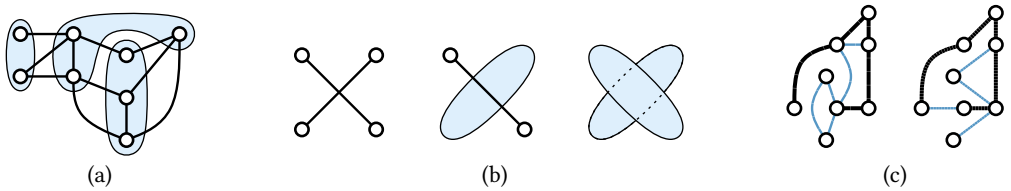


Figure 1.3: (a) A c-planar drawing of a clustered graph with three clusters. (b) An edge crossing, an edge-region crossing, and a region crossing (from left to right). (c) Two different graphs on the same vertex set with a simultaneous planar drawing; the drawing of each graph is planar and the common graph (bold) is drawn the same in both.

the drawing of the tree in Figure 1.1 becomes worthless when omitting the labels and the grouping of the vertices. Another example are UML diagrams illustrating the structure of a software project. Such a diagram is much more helpful if it not only displays the classes and relations between them but also their hierarchical clustering imposed by packages.

Such graph-drawing problems that go beyond simply drawing a single graph of course require extended graph drawing models and an adaption of the optimization criteria. E.g., a natural way to visualize a clustered graph is a drawing of the graph together with a colored region for each cluster; see Figure 1.3a. The optimization criterion of minimizing the number of edge crossings can be adapted to this extended graph-drawing model by minimizing crossings between pairs of edges, between edges and regions, and between pairs of regions; see Figure 1.3b. This also extends the fundamental concept of planarity (recall that a graph is planar if it can be drawn without edge crossings) to the concept of *clustered planarity* (or *c-planarity* for short) [Len89; FCE95b], where a clustered graph is c-planar if it admits a drawing with no crossings, i.e., without edge crossings, without edge-region crossings, and without region crossings. This leads to the problem Clustered Planarity of recognizing c-planar clustered graphs.

A similar extension to the concept of planarity is *simultaneous planarity*. Given two (or more) graphs that share some vertices and edges, simultaneous planarity asks whether each has a planar drawing such that the drawings of their common parts coincide; see Figure 1.3c. The problem of recognizing simultaneously planar graphs is usually called Sefe (which is an acronym for "simultaneous embedding with fixed edges") [EK05]. Simultaneously drawing multiple graphs is useful for comparing different graphs. E.g., when examining a dynamic network that changes over time, the user may be interested not only in its structure at particular points in time but also in understanding how the network changes.

The concepts of clustered and simultaneous planarity generalize the concept of planarity by allowing restrictions that go beyond requiring non-crossing edges. Generalizations of this type can be summarized using the common term *constrained planarity*.

1.2 Scope of the Thesis

Despite their long history and fundamental relevance, the computational complexity of many classical graph-drawing problems is unknown, i.e., neither efficient algorithms solving them nor proofs showing NP-hardness are known. The goal of this thesis is to advance the state of research on such classical graph-drawing problems. I focus on developing efficient algorithms. Whenever I prove a certain problem to be NP-hard, I oppose this negative result with a positive result. To this end, I for example develop parameterized algorithms whose running time increases exponentially with respect to a certain parameter but only polynomially in the input size.

The reader should not, however, expect ready-to-use algorithms for specific applications. The aim lies more in the development of a theoretical basis, providing techniques that can serve as a toolbox when actually implementing an algorithm for a certain application. Moreover, a better understanding of a problem's underlying mathematical properties facilitates the development of good heuristical algorithms in case techniques providing exact solutions cannot be applied.

1.3 Contribution and Outline

This thesis consists of two parts. First, I consider various variants of the bend-minimization problem for orthogonal drawings in different drawing models; see Part I. Then I investigate the constrained planarity problems Clustered Planarity and Sefe; see Part II.

1.3.1 Bend Minimization in Orthogonal Drawings

In orthogonal drawings, vertices are mapped to grid points and edges are routed along grid lines. As an edge can enter a vertex only from one of the four directions top, bottom, left, and right, each vertex can be connected to at most four neighbors, i.e., the degree of the vertices is bounded by 4. Thus, orthogonal drawings are often only considered for graphs with maximum degree 4. An extension of the basic orthogonal drawing style that is capable handling vertices of higher degree is the so-called Kandinsky model.

Graphs of Maximum Degree 4

Despite three decades of research on orthogonal drawings, many questions remained unanswered for a long time. It is for example known since 1994 that every *4-planar graph* (planar graph with maximum degree 4) admits an orthogonal drawing with at most two bends per edge (with a single exception) [BK98]. Concerning the computational complexity, this means that deciding whether a graph admits a two-bend

drawing is trivial. On the other hand, it is NP-hard to decide whether a 4-planar graph can be drawn without bends, i.e., whether it has a zero-bend drawing [GT01] (which is also known since 1994). This leaves open a huge complexity gap in the sense that going from two bends per edge to zero bends makes a trivial problem NP-hard. We partially closed this gap in 2010 by giving an algorithm that can efficiently decide whether a 4-planar graph admits a one-bend drawing, i.e., a drawing with one bend per edge [Blä+14].

Graphs with Inflexible Edges. In this thesis, I close this gap further by investigating the case where only some edges are *inflexible*, i.e., only some edges are required to have zero bends and all other edges can bend at least once; see Chapter 2. I develop an algorithm for testing whether a given graph with a set of inflexible edges admits such an orthogonal drawing. Its running time is exponential in the number of inflexible edges but polynomial in the size of the graph. More precisely, it has the running time $O(2^k \cdot n \cdot T_{\text{flow}}(n))$, where k is the number of inflexible edges and $T_{\text{flow}}(n)$ is the time necessary to compute a maximum flow in a planar flow network with multiple sources and sinks. This shows that the problem is *fixed-parameter tractable (FPT)* with respect to the number of inflexible edges. In fact, the running time is polynomial if the number of inflexible edges lies in $O(\log n)$, which is a stronger statement than being FPT. This is further strengthened by showing that it suffices to count only those inflexible edges that are incident to a vertex of degree 4.

On the other hand, I show that testing whether a 4-planar graph with inflexible edges admits an orthogonal drawing is NP-hard even if the graph contains only $O(n^\varepsilon)$ inflexible edges (for any constant $\varepsilon > 0$) that are evenly distributed over the graph, i.e., they have pairwise distance $\Omega(n^{1-\varepsilon})$. This includes the case where the inflexible edges form a matching in the graph.

Minimizing the Bends. The problems mentioned so far are decision problems, i.e., one has to decide whether a given graph has a drawing with the desired properties. The algorithms are usually constructive in the sense that a drawing is returned if one exists. However, if no drawing with the required properties exists, no drawing can be returned, which is usually not in the user's interest. E.g., if no zero-bend drawing exists, the user would like to have a drawing with as few bends as possible instead. Finding a bend-minimal drawing is of course NP-hard as the base case of deciding whether it can be done without bends is already hard.

However, testing the existence of a one-bend drawing can be done in polynomial time. Thus, one can hope for an efficient algorithm that minimizes the number of bends that go beyond the first bend on every edge. In Chapter 3, I prove several structural properties of orthogonal drawings that in fact lead to a polynomial-time algorithm that generates a bend-minimal orthogonal drawing when not counting the

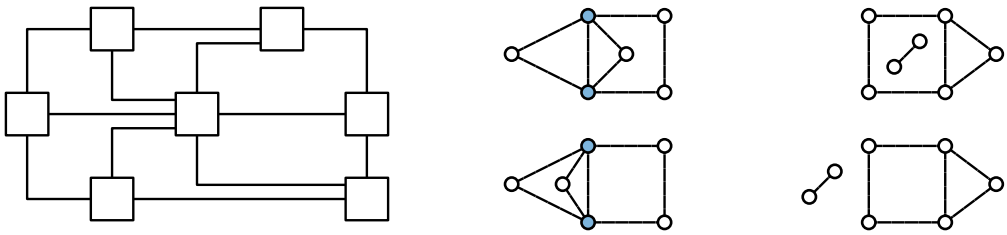


Figure 1.4: (a) An orthogonal drawing in the Kandinsky model. (b) Two planar drawings of the same graph. Their embedding is different as the edges incident to the blue vertices are ordered differently. (c) Two drawings of the same graph consisting of two connected components. The two drawings have different embeddings as the relative positions of the connected components are different.

first bend on every edge. The algorithm extends to the case where each edge has an individual cost function with the following two properties. First, each cost function must be convex. Second, the first bend on every edge must not cause any cost. The algorithm is optimal in the sense that omitting one of these two properties makes the problem NP-hard.

Kandinsky Drawings

To allow vertices with a degree greater than 4, the Kandinsky model represents vertices as squares of equal size, allowing multiple edges to enter a vertex at the same side; see Figure 1.4a. The Kandinsky model was introduced in 1995 [FK95] together with an efficient bend-minimization algorithm for the case that the input graph has a fixed embedding, i.e., the order of edges around every vertex is given; see Figure 1.4b. Unfortunately, this algorithm turned out to be flawed [Eig03]. Since then, the computational complexity of bend minimization in the Kandinsky model is open.

In spite of this, the Kandinsky model as such has received much attention, and different extensions to the model have been proposed, e.g., an extension to non-planar graphs [FK97] or to the case where vertices are represented by rectangles of prescribed size instead of uniform squares [Di +99]. Moreover, many approximation or heuristic algorithms for generating Kandinsky drawings with few bends have been developed.

In Chapter 4, I prove that the bend minimization problem in the Kandinsky model is NP-hard. On the positive side, I consider the problem's parameterized complexity with respect to different parameters. Among others, this results in a polynomial-time algorithm for series-parallel graphs (running time $O(n^3)$) and in a subexponential algorithm for general graphs (running time $2^{O(\sqrt{n}\log n)}$). Hence, I do not only answer a question that was open for almost twenty years, but also develop a new algorithmic approach for this NP-hard problem.

1.3.2 Constrained Planarity

In Part II of this thesis, I consider the constrained planarity problems Clustered Planarity and Sefe, both having an unknown computational complexity. Although these two problems arise from different applications and seem very different on the surface, they are actually very similar. In fact, Clustered Planarity (besides several other constrained planarity problems) can be reduced to Sefe [Sch13] and a slightly restricted case of Sefe can be reduced to Clustered Planarity [AL14]. Thus, an efficient algorithm for one of the two problems would also solve the other at least partially.

Besides giving new efficient algorithms solving restricted cases of Clustered Planarity and Sefe, I deepen the understanding of the connection between these two problems by showing that similar techniques can help to solve them. Note that this does not obviously follow from the above-mentioned reductions. Otherwise, all NP-complete problems could be approached using the same techniques just because they can be reduced to each other.

Before actually considering clustered and simultaneous planarity, I give a short introduction to the techniques centering around the problem Simultaneous PQ-Ordering in Chapter 5. These techniques will prove to be very useful in several subsequent chapters.

C-Planarity

Recall that a c-planar drawing of a clustered graph is a planar drawing of the graph together with a representation of the clusters with non-crossing regions such that no edge crosses a region; see Figure 1.3a. The problem Clustered Planarity was considered already in 1986 [Len89]. Despite numerous papers giving polynomial-time algorithms solving special cases of Clustered Planarity, the computational complexity of the general problem is still unknown.

In Chapter 6, I present a new data structure called cd-tree based on which I characterize the clustered graphs that are c-planar, providing a new perspective on c-planarity. By reconsidering previous results on c-planarity using this new perspective, I can show that many restricted cases that seem to be unnatural are actually very natural. This leads to a unification, often to a simplification, and in some cases to an extension of existing efficient algorithms. In particular, I give efficient algorithms solving Clustered Planarity for the cases that every cluster has at most five outgoing edges, or that every cluster and the complement of every cluster has at most two connected components.

Beyond the results presented in Chapter 6, the new perspective gives rise to several promising directions for further research on the topic. I would like to stress that the strength of the new perspective based on the cd-tree lies in its simplicity. This is

reflected in the fact that Chapter 6 is comparatively short even though it addresses many variants of c-planarity.

Simultaneous Planarity

Recall that the problem Sefe asks whether two given graphs that share a common subgraph admit planar drawings that coincide on the common subgraph. The current state of research on the problems Sefe and Clustered Planarity is very similar: There are many efficient algorithms for restricted cases but the computational complexity for the general case is unknown.

Besides other restrictions, most previous results assume that the common graph is connected. This simplifies Sefe insofar as it suffices to find planar drawings of the two input graphs such that the cyclic order of common edges around common vertices is consistent; see Figure 1.4b. If the common graph has multiple connected components, one additionally has to ensure that the relative positions of these connected components with respect to each other are consistent; see Figure 1.4c.

In Chapter 7, I approach Sefe from the opposite direction, considering cases where one only has to ensure consistent relative positions as the cyclic edge orderings do not matter. I solve this problem by giving an efficient algorithm for Sefe for the case that the embedding of each connected component of the common graph is fixed.

In Chapter 8, I combine the techniques for consistent relative positions from Chapter 7 with existing and newly developed methods for ensuring consistent edge orderings. This leads to an efficient algorithm solving Sefe if every connected component of the common graph is biconnected, has maximum degree 3, or is outerplanar with cutvertices of maximum degree 3. In the case that each connected component is biconnected, the algorithm even runs in optimal linear time. Besides our previous algorithm [BR13] that efficiently solves Sefe if the common graph is connected while both input graphs are biconnected, the algorithm presented in Chapter 8 is one of the least restrictive algorithms for Sefe in the sense that it can solve a large variety of instances.

1.4 Preliminaries

The purpose of this section is threefold. The first aim is to introduce the reader to the notation we[1] use, e.g., how we denote the degree of a vertex or how we distinguish between directed and undirected edges. Note that the chapter does not intend to be an introduction to fundamental graph-theoretic concepts [Die10], to basic algorithms and algorithmic tools such as run-time analysis [Cor+09a], to complexity-theoretic

[1] For the purpose of more convenient writing, I use "we" instead of "I" in the remainder of my thesis.

foundations like the notion of NP-completeness [GJ79], or to fixed-parameter tractability [DF13]. The second aim is to introduce the reader to less fundamental techniques that we use in multiple chapters of this thesis, such as SPQR-trees and orthogonal representations. The third aim is to give formal definitions for the different types of graph drawings and the corresponding graph-drawing problems considered in this thesis.

1.4.1 Graph-Theoretic Notation

Let $G = (V, E)$ be a graph with vertex set V and edge set E. With $n = |V|$ and $m = |E|$, we denote the number of vertices and edges, respectively. If not otherwise mentioned, we assume G to be undirected and simple, i.e., G has no multiple edges and no (self-) loops. Thus, E is a set of two-element subsets of V. For an edge $e = \{u, v\}$ we also use the short form uv (which is equal to vu). We say that u and v are the *endvertices* or *endpoints* of e and that e *connects* the *neighbors* u and v. The vertices u and v are *incident* to e and *adjacent* to each other. The *degree* of a vertex v is the number of incident edges and is denoted by $\deg(v)$.

The usual set terminology carries over from the vertices and edges to the graph. E.g., the union of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. Similarly, the graph $G' = (V', E')$ is a *subgraph* of G ($G' \subseteq G$) if $V' \subseteq V$ and $E' \subseteq E$. The subgraph G' is *induced* by V' if $E' = \{uv \in E \mid u \in V' \text{ and } v \in V'\}$. The subgraph G' is *proper* if $G' \neq G$ and $V' \neq \emptyset$. The subgraph obtained from G by removing a vertex v, i.e., the subgraph induced by $V \setminus \{v\}$, is also denoted by $G - v$. Similarly, G without the edge uv is denoted by $G - uv$ and G with an additional edge connecting the vertices u and v is denoted by $G + uv$.

A *walk* in a graph G is a sequence $(v_0, e_1, v_1, e_2, \ldots, v_{k-1}, e_k, v_k)$ such that v_i is a vertex for $i = 0, \ldots, k$ and e_i is an edge connecting v_{i-1} and v_i for $i = 1, \ldots, k$. A vertex or an edge may be contained multiple times in a walk. As the vertices of a walk already determine the edges (at least in a simple graph), we usually omit the edges in the above notation. The *length* of a walk $(v_0, \ldots, v_k)$ is k; the walk $(v_0, \ldots, v_k)$ is *cyclic* if $v_0 = v_k$. A walk is a *path* if no vertex appears multiple times in $(v_0, \ldots, v_k)$. A cyclic walk is a *cycle* if the $v_1, \ldots, v_k$ are distinct vertices. We also call the subgraph of G given by the vertices and edges of a path (or cycle) a path (or cycle) in G. A path $(u = v_0, \ldots, v_k = v)$ is also called *uv-path*.

When dealing with directed graphs, we also call the edges *arcs*. For an arc $a = (u, v)$, u is the *source* and v is the *target*. We say that a is an *outgoing* arc for u and an *incoming* arc for v. A walk $W = (v_0, e_1, v_1, e_2, \ldots, v_{k-1}, e_k, v_k)$ in a directed graph is *directed* if e_i is directed from v_{i-1} to v_i for $i = 1, \ldots, k$, i.e., $e_i = (v_{i-1}, v_i)$. This definition directly extends to directed paths and directed cycles. A directed graph without directed cycles is called *DAG (directed acyclic graph)*.

When G has multiple edges, we say that G is a *multi-graph*. We also use the prefix

multi- for restricted graph classes with multiple edges, e.g., a multi-cycle is a multi-graph such that removing multiple edges results in a cycle. Besides few exceptions, graphs we consider have no loops.

A graph G is *connected*, if G is not the disjoint union of two proper subgraphs. A *separating k-set* is a set of k vertices whose removal disconnects G. Separating 1-sets and 2-sets are *cutvertices* and *separating pairs*, respectively. A connected graph is *biconnected* if it has no cutvertex and a biconnected graph is *triconnected* if it has no separating pair. The maximal biconnected components (with respect to inclusion) of a graph are called *blocks*. Let S be a separating k-set of G. The *split components* with respect to S are the smallest subgraphs $G_1, \dots, G_\ell$ of G such that $G = G_1 \cup \dots \cup G_\ell$ and $G_i \cap G_j \subseteq S$ for every $i, j \in \{1, \dots, \ell\}$ with $i \neq j$. Note that if $\{s,t\}$ is a separating pair and st is an edge of G, then the graph consisting only of the edge st is one of the split components. A *split pair*, which can be seen as a generalization of separating pair, is a set of two vertices $\{s,t\}$ such that $\{s,t\}$ is a separating pair or st is an edge. The above definition of split components extends to the case where $\{s,t\}$ is a split pair but not a separating pair, in which case st and $G - st$ are the only two split components.

A *tree* is a connected graph that contains no cycle as subgraph. The vertices of a tree T with degree 1 are called *leaves*, vertices with higher degree are *inner vertices*. The tree T is *rooted* if it is directed such that each vertex has exactly one incoming arc except for one vertex with no incoming arcs. The latter vertex is the *root* of T. For an arc $a = (u,v)$ of T, the source u is also called *parent* of v and conversely, v is a *child* of u.

Sometimes, we use graphs as auxiliary structures. In this case, the vertices often represent more than an element of a set, e.g., a vertex may represent a subgraph of another graph. We call these kinds of vertices also *nodes* to distinguish them from vertices that are nothing more than elements of a set. A prominent example is the SPQR-tree defined in Section 1.4.3.

1.4.2 Drawings and Planar Embeddings

Let $G = (V,E)$ be a graph. A *drawing* Γ of G into the real plane $\mathbb{R}^2$ maps each vertex v of G to a point $\Gamma(v)$ and each edge e of G to an open Jordan curve $\Gamma(e)$ with the following properties. For $u, v \in V$, the points $\Gamma(u)$ and $\Gamma(v)$ are distinct if $u \neq v$. For $uv \in E$, the endpoints of $\Gamma(uv)$ are $\Gamma(u)$ and $\Gamma(v)$ and for every vertex $w \in V$, the point $\Gamma(w)$ is not contained in the interior of $\Gamma(uv)$. Finally, for edges $e_1 \neq e_2$, $\Gamma(e_1) \cap \Gamma(e_2)$ is finite. A drawing onto the sphere S^2 is defined analogously.

To improve readability, we usually identify the drawings of vertices and edges with the corresponding vertices and edges, respectively. E.g., a *crossing* of two edges e_1 and e_2 is a point shared by the interior of e_1 and the interior of e_2. It is (and it will always be) clear from the context that we actually refer to $\Gamma(e_1)$ and $\Gamma(e_2)$ instead of e_1 and e_2 themselves. If e_1 and e_2 have a crossing, we say they *cross*.

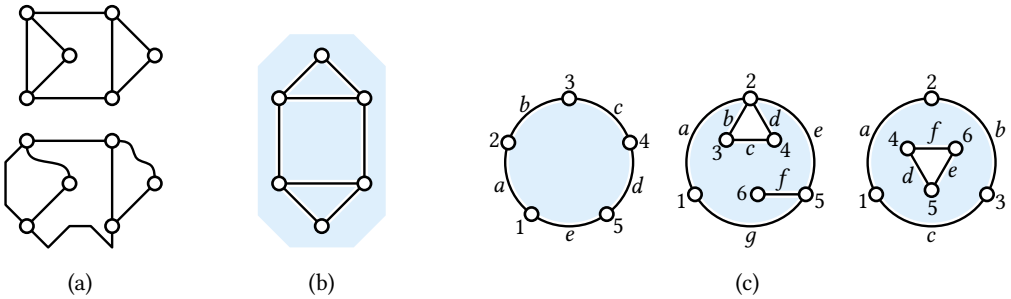


Figure 1.5: (a) Two drawings realizing the same planar embedding. (b) The faces of a drawing. (c) A face of a biconnected (left), connected (middle), and disconnected planar graph (right). The boundaries of these faces are the cycle $(1, a, 2, b, 3, c, 4, d, 5, e, 1)$ (left), the cyclic walk $(1, a, 2, b, 3, c, 4, d, 2, e, 5, f, 6, f, 5, g, 1)$ (middle), and a union of the two cyclic walks $(1, a, 2, b, 3, c, 1)$ and $(4, d, 5, e, 6, f, 4)$.

A crossing-free drawing of G is called *planar drawing*. A graph that has a planar drawing is *planar*. Note that planarity is independent from whether we draw in the plane or on the sphere as $\mathbb{R}^2$ and S^2 with a single point removed are homeomorphic and a drawing without crossings remains planar when applying a homeomorphism.

We define the following equivalence relation. Two planar drawings Γ_1 and Γ_2 of a planar graph G in the plane are equivalent if there exists a homeomorphism of the plane onto itself that maps Γ_1 to Γ_2; see Figure 1.5a. An equivalence class $\mathcal{E}$ with respect to this equivalence relation is called *planar embedding* of G. Analogously, we define *planar embedding on the sphere* for drawings on the sphere that can be transformed into one another by applying a homeomorphism of the sphere.

Let G be a planar graph together with a planar drawing Γ in the plane $\mathbb{R}^2$ (if not mentioned otherwise, all definitions are literally the same for drawings on the sphere S^2). Removing all edges from $\mathbb{R}^2$ (i.e., removing their image under Γ) may disconnect $\mathbb{R}^2$ into several connected components. These connected components are called *faces* of Γ; see Figure 1.5b. Exactly one of these faces is unbounded. It is called the *outer face*; all other faces are *inner faces*. This is not true for drawings on the sphere, where we have only inner faces. Let f be a face. If G is biconnected, the (topological) boundary of f corresponds to a cycle in G. We also call this cycle the *boundary* of f. When traversing the boundary of f, we usually choose an orientation such that the face f lies to the right of the boundary (i.e., we traverse f in clockwise direction if f is in inner face and in counter-clockwise direction if f is an outer face). If G is connected but not biconnected, the boundary of a face is a cyclic walk in which the same vertices and edges may appear several times. Finally, if G is disconnected, the boundary of a face is the union of cyclic walks; see Figure 1.5c. A vertex or an edge is *incident* to a face if it is contained in the boundary of this face. Two faces that are incident to the same edge are *adjacent*.

Note that two planar drawings Γ_1 and Γ_2 have the same planar embedding $\mathcal{E}$ on the sphere if and only if the boundaries of their faces are the same. This allows us to talk about faces of the planar embedding $\mathcal{E}$ instead of the faces of the planar drawings Γ_1 and Γ_2. If Γ_1 and Γ_2 are planar drawings in the plane, they have the same planar embedding if and only if they have the same faces and the same outer face.

A planar graph together with a fixed planar embedding is also called a *plane graph*. Let $G = (V, E)$ be a plane graph, and let F be the set of faces of G. The *dual graph* $G^\star = (F, E^\star)$ is defined as follows. The vertex set of $G^\star$ is the set of faces F and for every edge $e \in E$ incident to the faces f_1 and f_2, the set $E^\star$ includes the edge $e^\star = f_1 f_2$ *dual* to e. We also say that G is the *primal graph* of $G^\star$ and e is the *primal edge* of $e^\star$.

Why Drawing on the Sphere?

Usually, one is interested in drawings into the plane and not in drawings on the sphere. However, due to the fact that $\mathbb{R}^2$ and S^2 without a single point are homeomorphic, a drawing with certain properties on the sphere may imply the existence of a drawing with certain properties in the plane. E.g., a drawing with k crossings on the sphere implies the existence of a drawing with k crossings in the plane.

Drawing onto the sphere has the advantage that there is no special face such as the outer face in the plane, which often requires special attention. Thus, drawing onto the sphere lets us exploit symmetries that do not exist when drawing in the plane.

Using the sphere to break symmetries is, however, not always possible. E.g., the notion of an orthogonal drawing on the sphere does not exists. Moreover, the choice of the outer face plays an important role when generating planar orthogonal drawings. In Part I, we thus solely use drawings (and embeddings) in the plane. In Part II, we usually use the sphere when conceptually describing algorithms or proving structural results. However, when it comes to efficient implementations, it usually helps to have the outer face as a reference point.

Edge Orderings and Relative Positions

Let G be a graph with a planar drawing, and let v be a vertex. When traversing the edges incident to v in clockwise order, starting with an arbitrary edge, we get a linear order on the edges incident to v. This linear order depends on the edge we start with. Thus, the order in which the edges appear around v is better described by a so-called cyclic order, which is defined as follows.

Let $S = \{s_1, \dots, s_n\}$ be a finite set together with a linear order $s_1 < \cdots < s_n$. We denote this linear order also by $[s_1, \dots, s_n]$. The *cyclic shifts* of $[s_1, \dots, s_n]$ are the linear orders $[s_i, \dots, s_n, s_1, \dots, s_{i-1}]$ for $i = 1, \dots, n$. E.g., the order $[c, d, a, b]$ is a cyclic shift of $[a, b, c, d]$. Two linear orders are *cyclically equivalent* if one is the cyclic shift of the other. This relation is obviously an equivalence relation. The equivalence classes

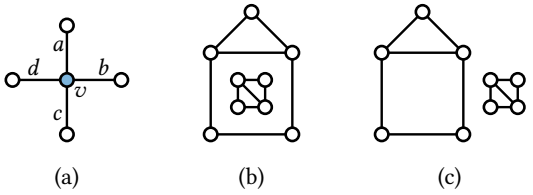


Figure 1.6: (a) A vertex v with edge ordering $[a, b, c, d]$. (b) A graph with two connected components. (c) The graph from (b) with different relative positions.

with respect to this relation are called *cyclic orders*. We denote a cyclic order using one representative of the equivalence class, e.g., $[a, b, c, d]$ and $[c, d, a, b]$ describe the same cyclic order.

Figure 1.6a shows a vertex v whose incident edges have the cyclic order $[a, b, c, d]$. We call the cyclic order in which the edges incident to a vertex v appear around v the *edge ordering* of v. Note that the edge orderings only depend on the embedding and not on the actual drawing. Conversely, in case G is connected, two drawings that have the same edge ordering for every vertex induce the same planar embedding on the sphere.

Assume G is not connected and let G_1 and G_2 be two connected components of G. In a planar drawing of G, the component G_2 lies in a single face f of G_1. We say that f is the *relative position* of G_2 with respect to G_1. Note that two drawings of G with the same edge ordering for every vertex may have different relative positions and thus different planar embeddings. E.g., the two drawings of the same graph in Figure 1.6b and Figure 1.6c have different embeddings although the embedding of each connected component and thus the edge orderings are the same. However, two drawings induce the same embedding on the sphere if and only if they have the same edge orderings and the same relative positions. They have the same embedding in the plane if they additionally have the same outer face.

Notation Conventions

We use *embedding* as a short term for planar embedding. In the literature, drawings of graphs are sometimes also called embedding. Throughout this thesis, we use solely the term drawing to have a clear distinction between drawings and planar embeddings.

Usually, it is clear from the context, whether we consider an embedding in the plane or on the sphere. Therefore, the term embedding can refer to a planar embedding or to a planar embedding on the sphere.

If we have fixed an embedding $\mathcal{E}$ for a planar graph G, we also write about the faces of G, which then actually refers to the faces of $\mathcal{E}$.

1.4.3 The SPQR-Tree

As embedding planar graphs involves choosing edge orderings for every vertex, planar graphs have potentially super-exponentially ($\Omega(n!)$) many planar embeddings. The

SPQR-tree is a data structure that compactly represents all planar embeddings of a biconnected planar graph [DT96a; DT96b]. It is based on a decomposition of biconnected graphs into triconnected components, which have a fixed planar embedding [Whi32]. Such a decomposition, and thus the SPQR-tree, can be computed in linear time [HT73; GM01]. A linear-time implementation [GM01] is available in the Open Graph Drawing Framework (OGDF) [Chi+13].

Composition of Graphs

An *st-graph* G is a graph with two designated vertices s and t such that $G + st$ is biconnected. The vertices s and t are called the *poles* of G. Let $G_1, \dots, G_k$ be st-graphs such that G_i (for $i = 1, \dots, k$) has the poles s_i and t_i. The *series composition* G of $G_1, \dots, G_k$ is the union of those graphs where t_i is identified with s_{i+1} for $i = 1, \dots, k-1$. Clearly, G is again an st-graph with the poles s_1 and t_k.

In the *parallel composition* G of $G_1, \dots, G_k$, the vertices $s_1, \dots, s_k$ and the vertices $t_1, \dots, t_k$ are merged into single vertices s and t, respectively. The vertices s and t are the poles of G. An st-graph is *series-parallel* if it is a single edge or the series or parallel composition of two series-parallel graphs.

To be able to compose all st-graphs, we need a third composition (not all st-graphs are series-parallel). Let $G_1, \dots, G_k$ be st-graphs. Moreover, let H be a graph with vertices s and t such that $H + st$ is triconnected and let $e_1, \dots, e_k$ be the edges of H. Then the *rigid composition* G with respect to the so-called *skeleton* $H + st$ is obtained by replacing each edge e_i of H with the graph G_i, identifying the endpoints of e_i with the poles of G_i. The special edge st is included in the skeleton for reasons that become clear later.

Every st-graph is either a single edge or the series, parallel or rigid composition of st-graphs. To make the notation more consistent, one can actually understand each of the three compositions as a replacement of edges in a skeleton $H + st$ with larger st-graphs. We get a series, parallel, and rigid composition if and only if the skeleton $H + st$ is a cycle, a bunch of (at least three) parallel edges, and a triconnected graph, respectively. See Figure 1.7 for an example. Note that the graph G_1 is decomposed in a tree-like fashion (each node represents a composition). The SPQR-tree is simply this decomposition tree. In the following sections, we give a formal definition of SPQR-trees, provide some useful notation, and discuss how the SPQR-tree represents all planar embeddings of a biconnected graph.

SPQR-Tree

An *SPQR-tree* $\mathcal{T}$ is a tree with the following properties. Every inner node of $\mathcal{T}$ is either an S-node, a P-node, or an R-node. With each inner node μ, we associate a graph $\text{skel}(\mu)$, the *skeleton* of μ. If μ is an S-, P-, or R-node, $\text{skel}(\mu)$ is a cycle, a bunch of at

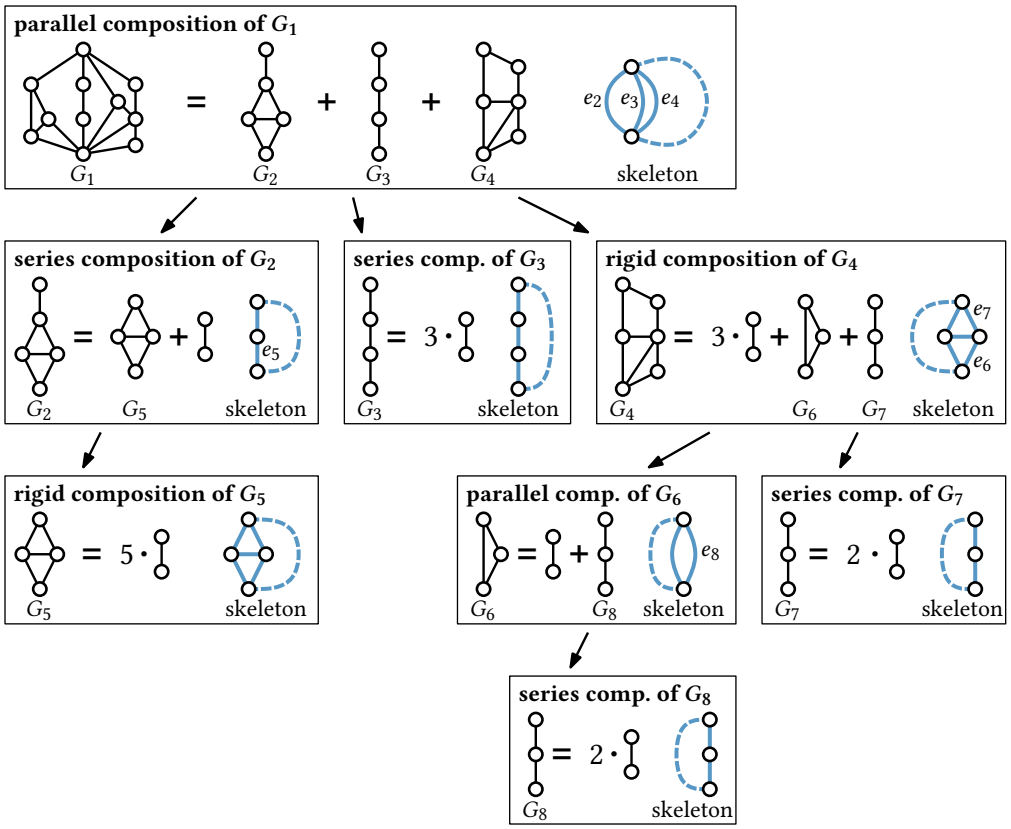


Figure 1.7: The *st*-graph G_1 is a parallel composition of G_2, G_3, and G_4. The graph G_2 is a series composition of an edge and G_5, which is a rigid composition of five edges. Similarly, G_3 and G_4 can be decomposed using only series, parallel, and rigid compositions.

least three parallel edges, and a triconnected graph, respectively. We call the edges in skel(μ) *virtual edges*. There is a bijection between the virtual edges of skel(μ) and the neighbors of μ in $\mathcal{T}$; we say that each virtual edge *corresponds* to a neighbor and vice versa. The leaves of $\mathcal{T}$ are Q-nodes (which exist mostly for technical reasons and to complete the name "SPQR-tree"). The skeleton of a Q-node μ is a pair of parallel edges, one of which is virtual (corresponding to the Q-node's unique neighbor). We say that μ *represents* the normal (i.e., non-virtual) edge of skel(μ).

Assume $\mathcal{T}$ to be rooted at a Q-node τ. For each node $\mu \neq \tau$, exactly one edge of skel(μ) corresponds to the parent of μ. We call it the *parent edge* and its endvertices the *poles* of μ. The poles of the root τ are the two vertices of skel(τ). Figure 1.8 shows an SPQR-tree $\mathcal{T}$. For each node μ of $\mathcal{T}$, we define the *pertinent graph* pert(μ) as follows. If μ is a Q-node (and not the root), pert(μ) is the edge represented by μ (i.e., the non-virtual edge of skel(μ)). If μ is an inner node with children $\mu_1, \ldots, \mu_k$,

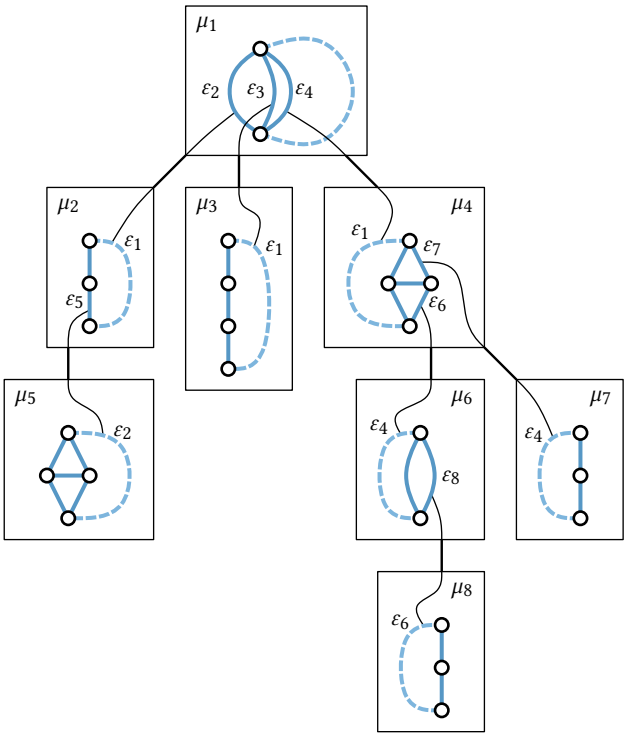


Figure 1.8: An SPQR-tree $\mathcal{T}$. The rectangles are the inner nodes $\mu_1, \ldots, \mu_8$ (the leaves are omitted in the drawing). The skeleton of each node is drawn inside its rectangle, the parent edge is dashed. For an edge ε_j in skel(μ_i), the node μ_j is the corresponding neighbor of μ_i (also indicated by the curved edges).

then pert(μ) is the composition of pert(μ_1), ..., pert(μ_k) with respect to the skeleton skel(μ), which is a series, parallel, and rigid composition if μ is an S-node, a P-node, and an R-node, respectively. Note that the pertinent graph pert(μ_i) in Figure 1.8 is the graph G_i in Figure 1.7.

The pertinent graph of the root τ of $\mathcal{T}$ is the pertinent graph of its unique child together with the edge represented by τ connecting its poles; let G be this graph. In Figure 1.8 the graph G is equal to $G_1 + st$, where G_1 is the graph from Figure 1.7. Note that choosing another root for $\mathcal{T}$ results in the same graph G. Moreover, if we require that no two S-nodes and no two P-nodes are adjacent in $\mathcal{T}$ (they would represent multiple successive series and parallel compositions, which can be replaced by a single series and parallel composition, respectively), there is no other SPQR-tree representing the same graph G. Thus, we can say that $\mathcal{T}$ is *the* SPQR-tree of G.

As all pertinent graphs are st-graphs, the graph G is biconnected. Moreover, every biconnected graph admits such a decomposition [HT73]. Thus, a graph has an SPQR-tree if and only if it is biconnected.

Representing Planar Embeddings

Let G be a biconnected graph and let $\mathcal{T}$ be its SPQR-tree. Then G is planar if and only if the skeleton of every node of $\mathcal{T}$ is planar. Moreover, making embedding choices for the skeletons carries over to the embedding of G. Which embedding choices we have for each skeleton depends on whether the SPQR-tree is unrooted, representing embeddings on the sphere, or rooted, representing embeddings in the plane.

For embeddings on the sphere, changing an edge ordering in one of the skeletons also changes the resulting edge orderings for G. We thus get a one-to-one correspondence between the embeddings of G (on the sphere) and the combinations of embeddings of the skeletons (also on the sphere). For each node μ of $\mathcal{T}$, this leads to the following embedding choices. If μ is an S-node or a Q-node, there are no choices as $\text{skel}(\mu)$ has a unique embedding. If μ is a P-node, we can choose any cyclic order for the edges of $\text{skel}(\mu)$. For a P-node of degree k, we get $(k-1)!$ different embeddings. If μ is an R-node, its embedding is fixed up to a *flip* (i.e., up to mirroring it by reversing all edge orderings) [Whi32].

For embeddings in the plane, things become less symmetric. Assume $\mathcal{T}$ is rooted at the Q-node τ and let s and t be the poles of τ. Then $\mathcal{T}$ can be used to represent all planar embeddings with the edge st on the outer face (if we want to represent all planar embeddings, we have to choose each Q-node as the root once). When composing graphs $G_1, \dots, G_k$, we can preserve their planar embeddings only if the embedding of G_i (for $i = 1, \dots, k$) has the poles of G_i on its outer face. Note that this is the case in the example of Figure 1.7. For every node $\mu \neq \tau$, we thus only allow planar embeddings of $\text{skel}(\mu)$ that have the parent edge on the outer face. Thus, our embedding choice for μ is to choose an embedding of $\text{skel}(\mu)$ on the sphere and pick one face incident to the parent edge as the outer face. Which of the two faces incident to the parent edge is chosen has no effect on the resulting planar embedding of G. Thus, for $\mu \neq \tau$, we actually get the same embedding choices as in the unrooted case (with a slightly different interpretation). For the root τ, $\text{skel}(\tau)$ has a unique embedding on the sphere (τ is a Q-node). However, τ has an embedding choice as choosing one of the two faces of $\text{skel}(\tau)$ as the outer face determines which of the faces incident to st is the outer face of G.

In the literature (and in the definitions above) it is usually required that the root τ of $\mathcal{T}$ is a Q-node. However, all definitions work exactly the same if the root τ is an inner node. Assume $\mathcal{T}$ to be rooted at an inner node τ. Also the embedding choices remain the same, including the fact that the choice of an outer face for $\text{skel}(\tau)$ has an effect on the outer face of G. E.g., if τ is an R-node, then we can choose one of the two embeddings of $\text{skel}(\tau)$ on the sphere and one of its (linearly many) faces as the outer face. As in the case where τ is a Q-node, this does not actually represent all planar embeddings of G, as the choice of the outer face is restricted. We have to choose multiple roots to actually get all planar embeddings.

Embeddings of Non-Biconnected Graphs

The SPQR-tree is only defined for biconnected graphs. However, if G is not biconnected, we can still use SPQR-trees to represent embeddings of the blocks of G. To this end, we define the *block-cutvertex tree (BC-tree)* $\mathcal{B}$ of a connected graph G as a tree whose nodes are the blocks and cutvertices of G, called *B-nodes* and *C-nodes*, respectively. In the BC-tree, a block B and a cutvertex v are connected by an edge if v belongs to B.

If an embedding is chosen for each block (represented by their SPQR-trees), these embeddings can be combined to an embedding of the whole graph if and only if $\mathcal{B}$ can be rooted at a B-node such that the parent of every non-root block B in $\mathcal{B}$, which is a cutvertex, lies on the outer face of B. Note that requiring a vertex v to lie on the outer face of a block B can be easily enforced by rooting the SPQR-tree of B only at Q-nodes corresponding to edges incident to v.

Notation

We introduce some additional notation that is not actually necessary but allows more concise writing. Let μ_1 be a node of $\mathcal{T}$ and let ε_2 be a virtual edge of μ_1. Moreover, let μ_2 be the neighbor of μ_1 corresponding to ε_2 and let ε_1 be the virtual edge of μ_2 that corresponds to μ_1; see Figure 1.8. We say that ε_2 is the *twin* of ε_1 and vice versa. We denote this relationship by $\operatorname{twin}(\varepsilon_1) = \varepsilon_2$.

Assume $\mathcal{T}$ is rooted at μ_1. Then the virtual edge ε_2 represents the pertinent graph $\operatorname{pert}(\mu_2)$. We say that this is the *expansion graph* of ε_2 and denote it by $\operatorname{expan}(\varepsilon_2)$. Note that the expansion graph is also defined for unrooted SPQR-trees, whereas the pertinent graph is not. A vertex in $\operatorname{expan}(\varepsilon)$ is an *inner vertex* if it is not an endvertex of ε.

In most cases we explicitly name the SPQR-trees we consider (e.g., by defining that $\mathcal{T}$ is the SPQR-tree of a given graph G). However, sometimes it is more convenient to write $\mathcal{T}(G)$ to denote the SPQR-tree of a given graph G.

We use the following conventions. With the SPQR-tree of a non-biconnected graph, we implicitly mean a collection of SPQR-trees, one for each block. For an S-, P-, Q-, or R-node μ of the SPQR-tree of a graph G, we also say that μ is an S-, P-, Q-, or R-node of G, respectively. These conventions for example simplify the statement "let μ be a P-node of the SPQR-tree of a block of G" to "let μ be a P-node of G".

1.4.4 Orthogonal Drawings

A drawing Γ of a graph G (in the plane) is *orthogonal* if it maps every edge of G to a sequence of alternating horizontal and vertical line segments; see Figure 1.9. The points on an edge where a horizontal and a vertical segment meet are called *bends*. One can assume that all vertices and bend points have integer coordinates. Thus, an orthogonal drawing is actually a drawing on the integer grid.

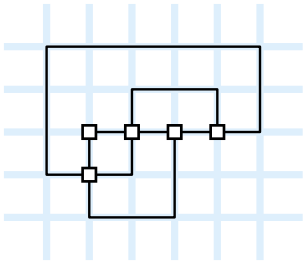


Figure 1.9: A planar orthogonal drawing of a 4-planar graph, i.e., a drawing on the (blue) grid. One edge has four bends, two edges have two bends, one edge has one bend, and all other edges have no bends.

For every vertex v of G, each edge incident to v enters v either from top, bottom, left, or right. Note that no two edges may enter v from the same side, as this would result in a pair of edges sharing an infinite number of points. It follows that orthogonal drawings exist only for graphs with maximum degree 4. A planar graph with maximum degree 4 is called *4-planar*.

We define the problem FlexDraw that includes multiple orthogonal drawing problems such as testing whether a graph has a drawing without bends or a drawing with one or two bends per edge. Let $G = (V, E)$ be a 4-planar graph together with a function $\text{flex}\colon E \to \mathbb{N}_0 \cup \{\infty\}$ assigning a *flexibility* to every edge. FlexDraw asks whether G admits an orthogonal drawing such that every edge $e \in E$ has at most $\text{flex}(e)$ bends. Such a drawing is called a *valid* drawing of the FlexDraw instance. An edge $e \in E$ with $\text{flex}(e) = 0$ is called *inflexible*. The instance has *positive flexibility* if no edge is inflexible.

The problem OptimalFlexDraw is the optimization problem corresponding to the decision problem FlexDraw. It is defined as follows. Let $G = (V, E)$ be a 4-planar graph together with a cost function $\text{cost}_e\colon \mathbb{N}_0 \to \mathbb{R} \cup \{\infty\}$ associated with every edge $e \in E$. The *cost* of an edge e with ρ bends in an orthogonal drawing of G is $\text{cost}_e(\rho)$. The cost of the whole drawing is the total cost summing over all edges. A drawing is *optimal* if it has the minimum cost among all orthogonal drawings of G. The task of the optimization problem OptimalFlexDraw is to find an optimal drawing of G.

Note that an instance of FlexDraw can be seen as an instance of OptimalFlexDraw in which the cost function of each edge e is set to $\text{cost}_e(\rho) = 0$ for $\rho \in [0, \text{flex}(e)]$ and $\text{cost}_e(\rho) = \infty$ for $\rho \in (\text{flex}(e), \infty)$. Deciding whether G admits a valid drawing is then equivalent to deciding whether it admits a drawing with cost less than ∞.

Thus, OptimalFlexDraw includes FlexDraw as a special case, which itself includes the problem of testing whether a 4-planar graph admits a drawing without bends. As the latter is NP-hard [GT01], FlexDraw and OptimalFlexDraw are both NP-hard problems. However, FlexDraw is efficiently solvable for instances with positive flexibility [Blä+14]. To obtain a similar result for OptimalFlexDraw in Chapter 3, we consider restricted cost functions.

For a cost function $\text{cost}_e(\cdot)$ we define the *difference function* $\Delta\,\text{cost}_e(\rho) = \text{cost}_e(\rho + 1) - \text{cost}_e(\rho)$. A cost function is *monotone* if its difference function is greater or

equal to 0. We say that the *base cost* of the edge e with monotone cost function is $b_e = \text{cost}_e(0)$. The *flexibility* of an edge e with monotone cost function is defined to be the largest possible number of bends ρ for which $\text{cost}_e(\rho) = b_e$. As before, we say that an instance G of OptimalFlexDraw has *positive flexibility* if all cost functions are monotone and the flexibility of every edge is positive.

The cost function $\text{cost}_e(\cdot)$ is *convex*, if its difference function is monotone. We call an instance of OptimalFlexDraw *positive-convex*, if every edge has positive flexibility and each cost function is convex. Note that this implies that the cost functions are monotone.

Orthogonal Representation

Two orthogonal drawings of a 4-planar graph G are *equivalent*, if they have the same topology, i.e., the same planar embedding, and the same shape in the sense that the sequence of right and left turns is the same in both drawings when traversing the faces of G. To make this precise, we define orthogonal representations as equivalence classes of this equivalence relation between orthogonal drawings. Orthogonal representations were first introduced by Tamassia [Tam87], however, we use a slight modification that makes it easier to work with, as bends of edges and bends at vertices are handled more consistently.

To ease the notation, we assume orthogonal drawings to be *normalized*, i.e., every edge has bends in only one direction. If additional bends do not improve the drawing (i.e., costs for bends are monotonically increasing), a normalized optimal drawing exists [Tam87]. Thus, assuming that orthogonal drawings are normalized is not a real restriction.

Let e be an edge in G that has β bends in Γ and let f be a face incident to e. We define the *rotation* of e in f to be $\text{rot}(e_f) = \beta$ and $\text{rot}(e_f) = -\beta$ if the bends of e form 90° and 270° angles in f, respectively. For a vertex v incident to a face f, we define $\text{rot}(v_f)$ to be 1, 0, −1 and −2 if v forms an angle of 90°, 180°, 270° and 360° in f, respectively. In other words, if v forms the angle α in f, we have $\text{rot}(v_f) = 2 - \alpha/90°$.

Note that, when traversing a face of G in clockwise direction (counter-clockwise for the outer face), the right and left bends correspond to rotations of 1 and −1, respectively (we may have two left bends at once at vertices of degree 1). The values for the rotations we obtain from a drawing Γ satisfy the following properties; see Figure 1.10a.

(1) The sum over all rotations in a face is 4 (−4 for the outer face).

(2) For every edge e with incident faces f_ℓ and f_r we have $\text{rot}(e_{f_\ell}) + \text{rot}(e_{f_r}) = 0$.

(3) The sum of rotations around a vertex v is $2 \cdot \deg(v) - 4$.

(4) The rotations at vertices lie in the range $[-2, 1]$.

Let $\mathcal{R}$ be a structure consisting of an embedding of G plus a set of values fixing the rotation for every vertex–face and edge–face incidence. We call $\mathcal{R}$ an *orthogonal representation* of G if the rotation values satisfy the above properties (1)–(4). Given an orthogonal representation $\mathcal{R}$, a drawing inducing the specified rotation values exists and can be computed efficiently [Tam87].

In some cases we also write $\text{rot}_{\mathcal{R}}(\cdot)$ instead of $\text{rot}(\cdot)$ to make clear which orthogonal representation we refer to. Moreover, the face in the subscript is sometimes omitted if it is clear which face is meant, e.g., we write $\text{rot}(e)$ instead of $\text{rot}(e_f)$.

We note that the above definitions are ambiguous if G is not biconnected. In this case an edge e or a vertex v may have multiple incidences to the same face f. Thus, it is not immediately clear, which angle is described by $\text{rot}(e_f)$ or $\text{rot}(v_f)$. However, it will be always clear from the context, which incidence of a face we refer to.

For a given instance of FlexDraw, we say that an orthogonal representation is *valid*, if a corresponding drawing is valid.

The Shape of *st*-Graphs

We extend the notion of rotation to paths; conceptually this is very similar to spirality [DLV98]. Let π be a path from vertex u to vertex v. We define the rotation of π (denoted by $\text{rot}(\pi)$) to be the number of bends to the right minus the number of bends to the left when traversing π from u to v.

There are two special paths in an st-graph G. Let s and t be the poles of G and let $\mathcal{R}$ be an orthogonal representation with s and t on the outer face. Then $\pi(s,t)$ denotes the path from s to t when traversing the outer face of G in counter-clockwise direction. Similarly, $\pi(t,s)$ is the path from t to s. We define the *number of bends* of $\mathcal{R}$ to be $\max\{|\text{rot}(\pi(s,t))|, |\text{rot}(\pi(t,s))|\}$. Note that a single edge $e = st$ is also an st-graph. Note further that the notions of the number of bends of the edge e and the number of bends of the st-graph e coincide. Thus, the above definition is consistent.

When considering orthogonal representations of st-graphs, we always require the poles s and t to be on the outer face. We say that the vertex s has σ *occupied incidences* if $\text{rot}(s_f) = \sigma - 3$ where f is the outer face. We also say that s has $4-\sigma$ *free incidences* in the outer face. If the poles s and t have σ and τ occupied incidences in $\mathcal{R}$, respectively, we say that $\mathcal{R}$ is a (σ,τ)-*orthogonal representation*; see Figure 1.10b.

Note that $\text{rot}(\pi(s,t))$ and $\text{rot}(\pi(t,s))$ together with the number of occupied incidences σ and τ basically describe the outer shape of G and thus how it has to be treated if it is a subgraph of some larger graph. Using the bends of $\mathcal{R}$ instead of the rotations of $\pi(s,t)$ and $\pi(t,s)$ implicitly allows to mirror the orthogonal representation (and thus exchanging $\pi(s,t)$ and $\pi(t,s)$).

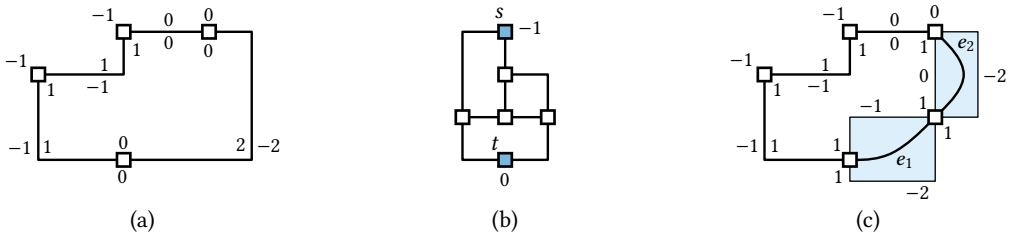


Figure 1.10: (a) An orthogonal drawing together with its orthogonal representation given by the rotation values. (b) A $(2,3)$-orthogonal representation (s and t have 2 and 1 free incidences, respectively). (c) An orthogonal representation with thick edges e_1 and e_2. The blue boxes indicate how many attachments the thick edges occupy, i.e., e_1 is a $(2,3)$-edge and e_2 is a $(2,2)$-edge. Both thick edges have two bends.

Thick Edges

In the basic formulation of an orthogonal representation, every edge *occupies* exactly one incidence at each of its endpoints, i.e., an edge enters each of its endpoint from exactly one of four possible directions. We introduce *thick edges* that may occupy more than one incidence at each endpoint to represent larger subgraphs.

Let $e = st$ be an edge in G. We say that e is a (σ,τ)-*edge* if e is defined to occupy σ and τ incidences at s and t, respectively. Note that the total amount of occupied incidences of a vertex in G must not exceed 4. With this extended notion of edges, we define a structure $\mathcal{R}$ consisting of an embedding of G plus a set of values for all rotations to be an *orthogonal representation* if it satisfies the following (slightly extended) properties; see Figure 1.10c.

(1) The sum over all rotations in a face is 4 (-4 for the outer face).

(2) For every (σ,τ)-edge e with incident faces f_ℓ and f_r we have $\operatorname{rot}(e_{f_\ell}) + \operatorname{rot}(e_{f_r}) = 2 - (\sigma + \tau)$.

(3) The sum of rotations around a vertex v with incident edges $e_1,\dots,e_\ell$ occupying $\sigma_1,\dots,\sigma_\ell$ incidences of v, respectively, is $\sum(\sigma_i + 1) - 4$

(4) The rotations at vertices lie in the range $[-2,1]$.

Note that requiring every edge to be a $(1,1)$-edge in this definition of an orthogonal representation exactly yields the previous definition without thick edges. The *number of bends* of a (thick) edge e incident to the faces f_ℓ and f_r is $\max\{|\operatorname{rot}(e_{f_\ell})|, |\operatorname{rot}(e_{f_r})|\}$. Unsurprisingly, replacing a (σ,τ)-edge with β bends in an orthogonal representation with a (σ,τ)-orthogonal representation with β bends of an arbitrary st-graph yields an orthogonal representation [Blä+14, Lemma 5].

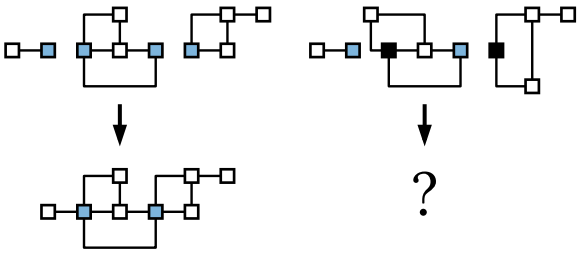


Figure 1.11: On the left three tight orthogonal drawings are stacked together. This is not possible on the right side, since the black vertices have angles larger than 90° in internal faces.

Tight Orthogonal Representations

Let G be a 4-planar graph with positive flexibility and valid orthogonal representation $\mathcal{R}$ and let $\{s,t\}$ be a split pair. Let further H be a split component with respect to $\{s,t\}$ such that the orthogonal representation $\mathcal{S}$ of H induced by $\mathcal{R}$ has $\{s,t\}$ on the outer face f. The orthogonal representation $\mathcal{S}$ of H is called *tight* with respect to the vertices s and t if the rotations of s and t in internal faces are 1, i.e., s and t form 90°-angles in internal faces of H.

Bläsius et al. [Blä+14, Lemma 2] show that $\mathcal{S}$ can be made tight with respect to s and t, i.e., there exists a valid orthogonal representation of H that is tight. Moreover, this tight orthogonal representation can be plugged back into the orthogonal representation of the whole graph G. We call an orthogonal representation $\mathcal{R}$ of the whole graph G *tight*, if every split component having the corresponding split pair on its outer face is tight with respect to its split pair. For instances with positive flexibility, we can thus assume without loss of generality that every valid orthogonal representation is tight.

Dealing with tight orthogonal representations has two major advantages. First, if we have for example a series composition of multiple st-graphs, each with a given orthogonal representation, we can easily combine these orthogonal representations yielding an orthogonal representation of the resulting graph (assuming it has maximum degree 4); see Figure 1.11. Note that this may not be possible if the orthogonal representations are not tight. Second, the shape of the outer face f of a split component with split pair $\{s,t\}$ is completely determined by the rotation of $\pi_f(s,t)$ and the degrees of s and t, since the rotation at the vertices s and t in the outer face only depends on their degrees. If not otherwise mentioned, we assume orthogonal representations of instances with positive flexibility to be tight.

Network Flows

A *flow network* is a tuple $N = (V, A, \text{COST}, \text{dem})$ where (V,A) is a directed (multi-) graph, COST is a set containing a *cost function* $\text{cost}_a \colon \mathbb{N}_0 \to \mathbb{R} \cup \{\infty\}$ for each arc $a \in A$ and $\text{dem} \colon V \to \mathbb{Z}$ is the *demand* of the vertices. A *flow* in N is a function $\phi \colon A \to \mathbb{N}_0$ assigning a certain amount of flow to each arc. A flow ϕ is *feasible*, if the difference of

incoming and outgoing flow at each vertex equals its demand, i.e.,

$$\mathrm{dem}(v) = \sum_{(u,v)\in A} \phi(u,v) - \sum_{(v,u)\in A} \phi(v,u) \quad \text{for all } v \in V.$$

The *cost* of a given flow ϕ is the total cost of the arcs caused by the flow ϕ, i.e.,

$$\mathrm{cost}(\phi) = \sum_{a\in A} \mathrm{cost}_a(\phi(a)).$$

A feasible flow ϕ in N is called *optimal* if $\mathrm{cost}(\phi) \leq \mathrm{cost}(\phi')$ holds for every feasible flow ϕ'.

If the cost function of an arc a is 0 on an interval $[0, c(a)]$ and ∞ on $(c(a), \infty)$, we say that a has *capacity* $c(a)$. If all arcs of N have such a cost function, we also write $N = (V, A, c, \mathrm{dem})$ instead of $N = (V, A, \mathrm{cost}, \mathrm{dem})$, where $c\colon A \to \mathbb{N}_0$ are the capacities.

Flows and Orthogonal Representations. Orthogonal representations have the following connection to flow networks. It was first discovered by Tamassia [Tam87]. Assume we have an orthogonal representation $\mathcal{R}$ for a graph G with planar embedding $\mathcal{E}$. The rotation of a vertex v in a face f is interpreted as flow between v and f, where a positive and negative rotation corresponds to flow from v to f and flow from f to v, respectively. Similarly, a positive and negative rotation of an edge e in a face f corresponds to flow from e to f and from f to e, respectively. Clearly, Property 1 of the orthogonal representation $\mathcal{R}$ (sum of all rotations in a face is 4) is equivalent to setting the demand of f to 4. Similarly, Property 2 requires the demand of edges to be 0 and by Property 3, the demand of vertices is $2 \cdot \deg(v) - 4$. That the rotations at vertices lie in the range $[-2, 1]$ (Property 4) can be enforced using capacities for the corresponding edges. Clearly, a feasible flow in the resulting network corresponds to an orthogonal representation with planar embedding $\mathcal{E}$ of G and every orthogonal representation of G with planar embedding $\mathcal{E}$ corresponds to a feasible flow in the network. Moreover, the number of bends an edge e has in the resulting orthogonal representation is equal to the flow going through e. Hence, one can use cost functions on the corresponding arcs to punish bends.

Modeling bend minimization in orthogonal drawings using this flow network only works if we know the faces of G, i.e., if the planar embedding of G is fixed. Nonetheless, we will use a similar flow network as a subroutine when optimizing over all planar embeddings.

Notation and Basic Properties of Flow Networks. In the following, we provide some more definitions related to flow networks and show basic properties we rely on in later chapters.

A flow network N is called *convex* if the cost functions on its arcs are convex. In the flow networks we consider, every arc $a \in A$ has a corresponding arc $a' \in A$ between the same vertices pointing in the opposite direction. A flow ϕ is *normalized* if $\phi(a) = 0$ or $\phi(a') = 0$ for each of these pairs. In a convex flow network, every optimal flow can be easily transformed to a normalized optimal flow. As all flow networks we consider are convex, we can assume all flows to be normalized.

We simplify the notation as follows. If we talk about an amount of flow on the arc a that is negative, we instead mean the same positive amount of flow on the opposite arc a'.

Many algorithms computing minimum-cost flows in networks can only handle linear cost functions, i.e., each unit of flow on an arc causes a constant cost defined for that arc. Note that the cost functions in a convex flow network N are piecewise linear and convex according to our definition. Thus, it can be easily formulated as a flow network with linear costs by splitting every arc into multiple arcs, each having linear costs. It is well known that flow networks of this kind can be solved in polynomial time [EK72]. When using an algorithm computing a minimum-cost flow in a flow network N as a subroutine, we usually denote the running time by $T_{\text{flow}}(|N|)$. This reflects the fact that finding a better algorithm for a certain flow problem also improves the running time of our algorithms. Applying for example the algorithm by Orlin [Orl93] leads to the running time $T_{\text{flow}}(n) \in O(n^2 \log^2 n)$, assuming that the number of edges of the flow network is linear in its number of vertices.

Let $u, v \in V$ be two nodes of the convex flow network N with demands $\text{dem}(u)$ and $\text{dem}(v)$. The *parameterized flow network* with respect to the nodes u and v is defined the same as N but with a *parameterized demand* of $\text{dem}(u) - \rho$ for u and $\text{dem}(v) + \rho$ for v where ρ is a parameter. The *cost function* $\text{cost}_N(\rho)$ of the parameterized flow network N is defined to be $\text{cost}(\phi)$ of an optimal flow ϕ in N with respect to the parameterized demands determined by ρ. Note that increasing ρ by 1 can be seen as pushing one unit of flow from u to v. We define the *optimal parameter* ρ_0 to be the parameter for which the cost function is minimal among all possible parameters. The correctness of the minimum weight path augmentation method to compute flows with minimum costs implies the following theorem [EK72].

Theorem 1.1. *The cost function of a parameterized flow network is convex on the interval* $[\rho_0, \infty)$*, where* ρ_0 *is the optimal parameter.*

Proof. Let $N = (V, A, \text{cost}, \text{dem})$ be a parameterized flow network and let ϕ_0 be a minimum-cost flow in N with respect to the optimal parameter ρ_0. To simplify notation, we assume $\rho_0 = 0$. The *residual network* R_0 with respect to ϕ_0 is the graph (V, A) with a constant cost $\text{cost}_0(a)$ assigned to every arc a such that $\text{cost}_0(a)$ is the amount of cost in N that has to be payed to push an additional unit of flow along a, with respect to the given flow ϕ_0. Note that this cost may be negative. It is well known that an optimal flow ϕ_1 with respect to the parameter 1 can be computed by

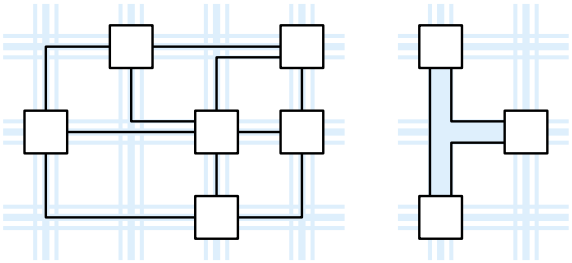
pushing one unit of flow along a path from u to v with minimum weight in R_0 [EK72]. Moreover, we can continue and compute an optimal flow ϕ_{k+1} by augmenting ϕ_k along a minimum weight path in the residual network R_k with respect to the flow ϕ_k. Assume we augment ϕ_k along the path π_k causing cost $\text{cost}_k(\pi_k)$ to obtain an optimal flow ϕ_{k+1} with respect to the parameter $k+1$ and then we augment along a path π_{k+1} in R_{k+1} with cost $\text{cost}_{k+1}(\pi_{k+1})$ to obtain an optimal flow ϕ_{k+2} with respect to the parameter $k+2$. To obtain the claimed convexity we have to show that $\text{cost}_k(\pi_k) \leq \text{cost}_{k+1}(\pi_{k+1})$ holds.

If π_k and π_{k+1} contain an arc a in the same direction, then $\text{cost}_k(a) \leq \text{cost}_{k+1}(a)$ holds by the convexity of the cost function of a. If π_k contains the arc a and π_{k+1} contains the arc a' in the opposite direction then $\text{cost}_k(a) = -\text{cost}_{k+1}(a')$ holds. Assume π_k and π_{k+1} share such an arc in the opposite direction. Then we remove this arc in both directions, splitting each of the paths π_k and π_{k+1} into two subpaths. We define two new paths π and π' by concatenating the first part of π_k with the second part of π_{k+1} and vice versa, respectively. This can be done iteratively, thus we can assume that π and π' do not share arcs in the opposite direction. We consider the cost of π and π' in the residual network R_k. Obviously, for an arc a that is exclusively contained either in π or in π' we have $\text{cost}_k(a) = \text{cost}_{k+1}(a)$. For an arc that is contained in π and π' we have $\text{cost}_k(a) \leq \text{cost}_{k+1}(a)$. Moreover, for every pair of arcs a and a' that was removed, we have $\text{cost}_k(a) = -\text{cost}_{k+1}(a')$. This yields the inequality $\text{cost}_k(\pi_k) + \text{cost}_{k+1}(\pi_{k+1}) \geq \text{cost}_k(\pi) + \text{cost}_k(\pi')$. Since π_k was a path with smallest possible weight in R_k we have $\text{cost}_k(\pi_k) \leq \text{cost}_k(\pi)$ and $\text{cost}_k(\pi_k) \leq \text{cost}_k(\pi')$. With the above inequality this yields $\text{cost}_{k+1}(\pi_{k+1}) \geq \text{cost}_k(\pi_k)$. □

In Chapter 4, we will need the following result on the existence of feasible flows in flow networks where the capacity of edges is large compared to the absolute demands of the nodes in network.

Lemma 1.1. *Let $N = (V, A, c, \text{dem})$ be a flow network with $\sum_{v \in V} \text{dem}(v) = 0$. If $c(a) \geq \sum_{v \in V} |\text{dem}(v)|$ holds for each arc $a \in A$, then there exists a feasible flow in N.*

Proof. Let ϕ be an arbitrary flow satisfying the demands at all vertices, but possibly violating the capacity constraints. Let $a = (u, v) \in A$ with $\phi(a) > c(a)$. If there exists a directed path from v to u all whose arcs have positive flow, we can decrease the amount of flow on this cycle by 1. After finitely many such steps, we then obtain the desired flow. Hence, assume for the sake of contradiction that such a path does not exist. Let $S \subseteq V$ be the vertices that can be reached from v. Note that $v \in S$ and $u \notin S$. Hence, S defines a cut in N whose outgoing arcs have flow 0. In any valid flow the amount of flow entering S minus the flow leaving S must equal $\sum_{v \in S} \text{dem}(v) \leq \sum_{v \in S} |\text{dem}(v)| \leq \sum_{v \in V} |\text{dem}(v)| \leq c(e)$. On the other hand, the flow entering S is at least $\phi(a) > c(e)$ while no flow is leaving S, a contradiction. □



(a) (b)

Figure 1.12: (a) A Kandinsky drawing of the wheel of size 5. (b) A Kandinsky drawing with an empty face.

1.4.5 Kandinsky Drawings

Let G be a planar graph. Recall that an orthogonal drawing of G maps each vertex to a grid point and each edge to a path in the grid; see Figure 1.9. Recall further that G has an orthogonal drawing only if it has maximum degree 4. The Kandinsky model introduced by Fößmeier and Kaufmann [FK95] is a way to overcome this limitation. A *Kandinsky drawing* of G maps each vertex to a box of constant size centered at a grid point and each edge to a path in a finer grid; see Figure 1.12a for an example. The problem of finding a Kandinsky drawing with the minimum number of bends is called Kandinsky Bend Minimization.

In a Kandinsky drawing, a face is *empty* if it does not include a grid cell of the coarser grid; see Figure 1.12b. Empty faces are empty in the sense that there is not enough space to add a vertex inside. Usually, one forbids empty faces in Kandinsky drawings as allowing empty faces requires a special treatment for faces of size 3 compared to larger faces and cycles that are no faces. We always assume that empty faces are forbidden except when explicitly allowing them.

Every Kandinsky drawing has the so called *bend-or-end property*, which can be stated as follows. One can declare a bend on an edge $e = uv$ to be *close* to v if it is the first bend when traversing e from v to u with the additional requirement that a bend cannot be close to both endpoints u and v. The bend-or-end property requires that an angle of $0°$ between edges uv and vw in the face f implies that at least one of the edges uv and vw has a bend close to v that is concave in f ($270°$ angle). Note that the triangle in Figure 1.12b does not have this property as the two concave bends cannot be close to all three vertices with $0°$ angles.

In contrast to Chapters 2 and 3, the graphs we consider in Chapter 4 have a fixed planar embedding, i.e., they are plane (for planar graphs with a variable embedding, Kandinsky Bend Minimization is known to be NP-hard [GT01]). When writing about an orthogonal or Kandinsky drawing of a plane graph, we of course require that this drawing respects the fixed embedding of the graph.

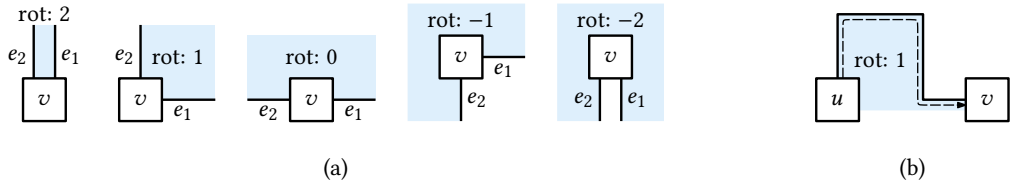


Figure 1.13: (a) The possible rotations at a vertex in the face f (shaded blue). (b) The rotation of an edge.

Kandinsky Representation

A Kandinsky drawing of a planar graph G can be specified in three stages. First, its topology is fixed by choosing a combinatorial embedding of G (which we assume to get with the input in case of Kandinsky drawings). Second, its shape in terms of angles between edges and sequences of bends on edges is fixed. Third, the geometry is fixed by specifying integer coordinates for all vertices and bend points. As with orthogonal representations for orthogonal drawings, we can define Kandinsky representations as the equivalence classes of Kandinsky drawings with the same topology and the same shape. As the number of bends (and thus the cost of a drawing) depends only on the shape and not on the geometry, we can focus on finding Kandinsky representations and thus neglect the geometry (at least if we make sure that every Kandinsky representation has a geometric realization as a Kandinsky drawing). As mentioned before, this approach was introduced for orthogonal drawings by Tamassia [Tam87]. It was extended to Kandinsky drawings by Fößmeier and Kaufmann [FK95].

The following definitions are very similar to the ones for orthogonal representations. However, we need slightly more notation as we can for example no longer assume drawings to be normalized, i.e., an edge may have bends in both directions.

Let G be a planar graph with the Kandinsky drawing Γ. Let f be a face with the edge e_1 in its boundary and let e_2 be the successor of e_1 in clockwise direction (counter-clockwise if f is the outer face). Let further v be the vertex between e_1 and e_2 and let α be the angle at v in f. We define the *rotation* $\mathrm{rot}_f(e_1, e_2)$ between e_1 and e_2 to be $\mathrm{rot}_f(e_1, e_2) = 2 - \alpha/90°$; see Figure 1.13a. The rotation $\mathrm{rot}_f(e_1, e_2)$ can be interpreted as the number of right turns between the edges e_1 and e_2 at the vertex v in the face f. Note that $e_1 = e_2$ if v has degree 1, which yields $\mathrm{rot}_f(e_1, e_2) = -2$. In case it is clear from the context which two edges are meant when referring to the vertex v in the face f, we also write $\mathrm{rot}_f(v)$ instead of $\mathrm{rot}_f(e_1, e_2)$ and call it the *rotation of v in f*.

The shape of every edge can also be described in terms of its rotation. Let u be a vertex in the boundary of the face f and let v be its successor in clockwise direction (counter-clockwise if f is the outer face). Let further $e = uv$ be the corresponding edge. The *rotation* $\mathrm{rot}_f(e)$ of e in f is the number of right bends minus the number of left bends one encounters, when traversing e from u to v; see Figure 1.13b. Note that every

edge has two rotations, one in each face it bounds. Note further, that our notation is not precise for bridges, as a bridge is incident to the same face twice. However, it will always be clear from the context which incidence is meant, hence there is no need to complicate the notation.

Let (u, v, w) be a path of length 2 in the face f. If the two edges form an angle of $0°$ (i.e., $\mathrm{rot}_f(v) = 2$), the bend-or-end property of Kandinsky drawings ensures that at least one of the two edges uv or vw has a bend close to v that forms an angle of $270°$ in f. To represent this information of which bends are declared to be close to vertices we introduce some additional rotations. Consider the edge uv and let f be an incident face. If uv has a bend close to v, we define the *rotation* $\mathrm{rot}_f(uv[v])$ *at the end* v of uv to be 1 if it is a right bend and -1 if it is a left bend. If uv has no bend close to v, we set $\mathrm{rot}_f(uv[v]) = 0$.

It is easy to see that every Kandinsky representation satisfies the following properties. Moreover, it is known that a set of values for the rotations is a Kandinsky representation if it satisfies these properties [FK95] (i.e., there exists a Kandinsky drawing with these rotation values).

(1) The sum over all rotations in a face is 4 (-4 for the outer face).

(2) For every edge uv with incident face f_ℓ and f_r, we have $\mathrm{rot}_{f_\ell}(uv) + \mathrm{rot}_{f_r}(uv) = 0$, $\mathrm{rot}_{f_\ell}(uv[u]) + \mathrm{rot}_{f_r}(uv[u]) = 0$, and $\mathrm{rot}_{f_\ell}(uv[v]) + \mathrm{rot}_{f_r}(uv[v]) = 0$.

(3) The sum of rotations around a vertex v is $2 \cdot \deg(v) - 4$.

(4) The rotations at vertices lie in the range $[-2, 2]$.

(5) If $\mathrm{rot}_f(uv, vw) = 2$ then $\mathrm{rot}_f(uv[v]) = -1$ or $\mathrm{rot}_f(vw[v]) = -1$.

If the face is clear from the context, we often omit the subscript in rot_f. Note that the rotation of an edge uv is split into three parts; the rotations $\mathrm{rot}(uv[u])$ and $\mathrm{rot}(uv[v])$ at the ends of uv and a *rotation* $\mathrm{rot}(uv[-])$ *in the center* of uv. It holds $\mathrm{rot}(uv) = \mathrm{rot}(uv[u]) + \mathrm{rot}(uv[-]) + \mathrm{rot}(uv[v])$ (thus it is not necessary to have $\mathrm{rot}(uv[-])$ contained in the representation). We can assume without loss of generality that all bends accounting for the rotation in the center bend in the same direction, thus the edge uv has $|\mathrm{rot}(uv[u])| + |\mathrm{rot}(uv[-])| + |\mathrm{rot}(uv[v])|$ bends. Hence, the number of bends depend only on the Kandinsky representation and not on the actual drawing.

Let f be a face of G and let u and v be two vertices on the boundary of f. By $\pi_f(u, v)$ we denote the path from u to v on the boundary of f in clockwise direction (counterclockwise for the outer face). As for orthogonal drawings, the rotation $\mathrm{rot}_f(\pi)$ of a path π in the face f is defined as the sum of all rotations of edges and inner vertices of π in f.

Note that an orthogonal drawing of a 4-planar graph is basically a Kandinsky drawing without $0°$ angles at vertices. It is thus not surprising that the definition of

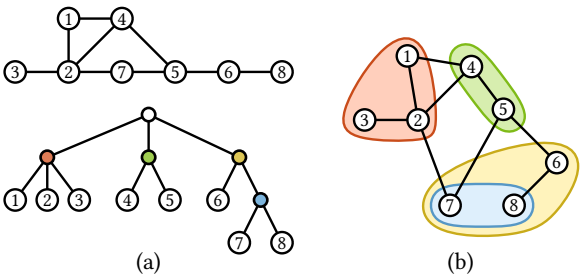


Figure 1.14: (a) A clustered graph consisting of a graph (top) and a hierarchical clustering of the vertices described by a rooted tree (bottom). (b) A c-planar drawing of the clustered graph in (a).

orthogonal representation from Section 1.4.4 is the same as the above definition of Kandinsky representation when forbidding 0° angles.

1.4.6 Clustered Planarity

A *clustered graph* (G,T) is a graph G together with a rooted tree T whose leaves are the vertices of G; see Figure 1.14a. Let μ be a node of T. The tree T_μ is the subtree of T consisting of all successors of μ together with the root μ. The graph induced by the leaves of T_μ is a *cluster* in G. We identify this cluster with the node μ. We call a cluster *proper* if it is neither the whole graph (root cluster) nor a single vertex (leaf cluster).

A *c-planar drawing* of (G,T) is a planar drawing of G in the plane together with a *simple* (= simply-connected) region R_μ for every cluster μ satisfying the following properties; see Figure 1.14b. (i) Every region R_μ contains exactly the vertices of the cluster μ. (ii) Two regions have non-empty intersection only if one contains the other. (iii) Edges cross the boundary of a region at most once. A clustered graph is *c-planar* if and only if it admits a c-planar drawing. The problem Clustered Planarity asks whether a clustered graph is c-planar.

The above definition of c-planarity relies on embeddings in the plane using terms like "outside" and "inside". Instead, one can consider drawings on the sphere by considering the tree T to be unrooted instead of rooted, using cuts instead of clusters, and simple closed curves instead of simple regions. Let ε be an edge in T. Removing ε splits T in two connected components. As the leaves of T are the vertices of G, this induces a *corresponding cut* $(V_\varepsilon, V'_\varepsilon)$ with $V'_\varepsilon = V \setminus V_\varepsilon$ on G. For a c-planar drawing of G on the sphere, we require a planar drawing of G together with a simple closed curve C_ε for every cut $(V_\varepsilon, V'_\varepsilon)$ with the following properties. (i) The curve C_ε separates V_ε from V'_ε. (ii) No two curves intersect. (iii) Edges of G cross C_ε at most once.

Note that using clusters instead of cuts corresponds to orienting the cuts, using one side as the cluster and the other side as the cluster's complement; the *co-cluster*. This notion of c-planarity on the sphere is equivalent to the one on the plane; one simply has to choose an appropriate point on the sphere to lie in the outer face. The unrooted view has the advantage that it is more symmetric (i.e., there is no difference

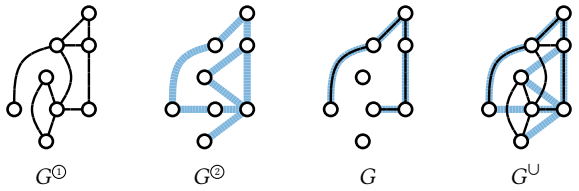


Figure 1.15: Two graphs $G^{①}$ and $G^{②}$, their common graph G, and their union graph $G^{\cup}$. The drawings of $G^{①}$ and $G^{②}$ are simultaneously planar as they are planar and coincide on G.

between clusters and co-clusters), which is sometimes desirable. We use the rooted and unrooted view interchangeably.

1.4.7 Simultaneous Planarity

Let $G^{①} = (V^{①}, E^{①})$ and $G^{②} = (V^{②}, E^{②})$ be two graphs. We call their intersection $G = G^{①} \cap G^{②}$ the *common graph* and their union $G^{\cup} = G^{①} \cup G^{②}$ the *union graph*. Drawings $\Gamma^{①}$ and $\Gamma^{②}$ of $G^{①}$ and $G^{②}$ form a *simultaneous planar drawing*, if $\Gamma^{①}$ and $\Gamma^{②}$ are both planar and coincide on the common graph G; see Figure 1.15. The problem Sefe has a pair of graphs $(G^{①}, G^{②})$ as input and asks whether they admit a simultaneous planar drawing.

Clearly, the definition directly extends to more than two graphs. A special case is the so-called *sunflower* case, in which the intersection graph is the same for every pair of input graphs. If not otherwise mentioned, we always consider the case of two graphs.

Edges belonging to the common graph G are called *common edges*, edges belonging to $G^{①}$ or $G^{②}$ but not to the common graph G are called *exclusive*.

Jünger and Schulz [JS09] showed that two graphs $G^{①}$ and $G^{②}$ admit a simultaneous planar drawing if and only if they have a *simultaneous embedding*, i.e., if they have planar embeddings that coincide on the common graph. Thus, solving Sefe for two graphs $G^{①}$ and $G^{②}$ boils down to finding embeddings of $G^{①}$ and $G^{②}$ that induce the same edge orderings (*consistent edge orderings*) and the same relative positions (*consistent relative positions*) on the common graph G.

Part I

Orthogonal Drawings

2 Inflexible Edges in Orthogonal Drawings

In this chapter, we investigate the problem FlexDraw in the presence of inflexible edges. I.e., we want to decide whether a given 4-planar graph admits a planar orthogonal drawing such that no edge e has more bends than allowed by its flexibility $\text{flex}(e)$. Recall that requiring positive flexibilities makes FlexDraw polynomial-time solvable [Blä+14], whereas it is NP-hard in the presence of inflexible edges [GT01], i.e., edges with flexibility 0.

To close the gap between the NP-hardness for $\text{flex}(e) = 0$ and the efficient algorithm for $\text{flex}(e) \geq 1$, we investigate the computational complexity of FlexDraw in case only few edges are inflexible. We show that for any $\varepsilon > 0$ FlexDraw is NP-complete for instances with $O(n^\varepsilon)$ inflexible edges of pairwise distance $\Omega(n^{1-\varepsilon})$ (including the case where they induce a matching). On the other hand, we give an FPT-algorithm with running time $O(2^k \cdot n \cdot T_{\text{flow}}(n))$, where $T_{\text{flow}}(n)$ is the time necessary to compute a maximum flow in a planar flow network with multiple sources and sinks, and k is the number of inflexible edges having at least one endpoint of degree 4. It is known that $T_{\text{flow}}(n) \in O(n \log^3 n)$ [Bor+11] for this type of flow problem.

This chapter is based on joint work with Sebastian Lehmann and Ignaz Rutter [BLR15].

2.1 Introduction

The problem of minimizing the number of bends in orthogonal drawings was first considered by Storer [Sto80] in 1980. However, the research on bend minimization is usually considered to be initiated by a result of Tamassia [Tam87] from 1984. Tamassia showed that bend minimization for plane graphs, i.e., for planar graphs with a fixed planar embedding, can be reduced to finding a minimum-cost flow in a planar flow network. Thus, OptimalFlexDraw can be solved efficiently if the planar embedding is fixed and the cost functions are convex. Cornelsen and Karrenbauer [CK12] recently showed that this kind of flow problem can be solved in $O(n^{3/2})$ time.

However, the restriction to a fixed planar embedding can have a huge impact on the number of bends; see Figure 2.1. Unfortunately, minimizing the number of bends over all planar embeddings is NP-hard, as shown by Garg and Tamassia [GT01]. In fact, they showed that it is already NP-hard to test whether a graph admits a drawing without bends. This is equivalent to the problem FlexDraw if all edges are inflexible.

On the positive side, Di Battista et al. [DLV98] showed that FlexDraw can be solved efficiently for series-parallel and maximum-degree-3 graphs. In fact, their algorithm

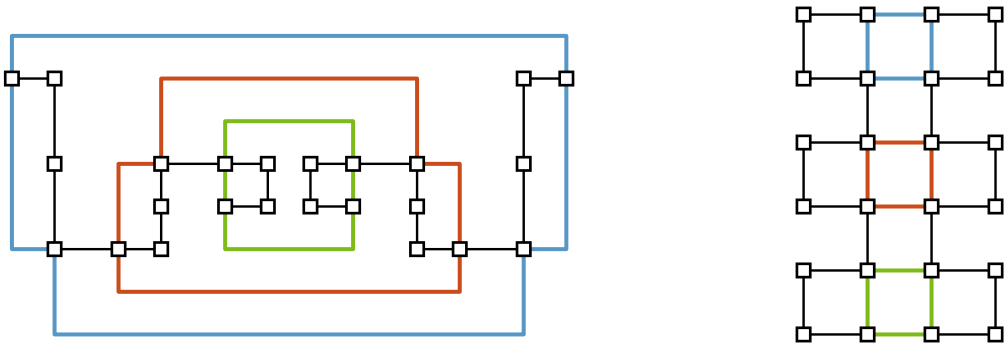


Figure 2.1: Two orthogonal drawings of the same graph with different planar embeddings. They are both bend-minimal with respect to their planar embeddings. Three cycles are colored to make a comparison of the two drawings easier.

can even do bend minimization, solving OptimalFlexDraw with $\text{cost}_e(\beta) = \beta$ for each edge e. Biedl and Kant [BK98] showed that every 4-planar graph (except for the octahedron) admits an orthogonal drawing with at most two bends per edge. Thus, from a complexity point of view, FlexDraw is trivial if the flexibility of every edge is at least 2. Bläsius et al. [Blä+14] gave a polynomial time algorithm solving FlexDraw if the flexibility of every edge is at least 1.

Contribution and Outline

In this chapter, we consider FlexDraw (and to a small degree also OptimalFlexDraw) for instances that may contain inflexible edges, closing the gap between the general NP-hardness result [GT01] and the polynomial-time algorithm in the absence of inflexible edges [Blä+14]. We do this, by strengthening both results, the NP-hardness proof if $\text{flex}(e) = 0$ and the polynomial-time algorithm if $\text{flex}(e) \geq 1$ for every edge e. The latter is extended into another direction in Chapter 3, where we give an efficient algorithm solving positive-convex instances of OptimalFlexDraw.

In Section 2.2, we show that FlexDraw remains NP-hard even for instances with only $O(n^\varepsilon)$ (for any $\varepsilon > 0$) inflexible edges that are distributed evenly over the graph, i.e., they have pairwise distance $\Omega(n^{1-\varepsilon})$. This includes the cases where the inflexible edges are restricted to form very simple structures such as a matching.

On the positive side, we describe a general algorithm that can be used to solve OptimalFlexDraw by solving smaller subproblems (Section 2.3). This provides a framework for the unified description of bend minimization algorithms which covers both, previous work and results presented in this paper. We use this framework in Section 2.4 to solve OptimalFlexDraw for series-parallel graphs with monotone cost functions. This extends the algorithm of Di Battista et al. [DLV98] by allowing

a significantly larger set of cost functions (in particular, the cost functions may be non-convex).

In Section 2.5, we present the main result of this chapter, which is an FPT-algorithm with running time $O(2^k \cdot n \cdot T_{\text{flow}}(n))$, where k is the number of inflexible edges incident to degree-4 vertices, and $T_{\text{flow}}(n)$ is the time necessary to compute a maximum flow in a planar flow network of size n with multiple sources and sinks. Note that we can require an arbitrary number of edges whose endpoints both have degree at most 3 to be inflexible without increasing the running time.

2.2 A Matching of Inflexible Edges

In this section, we show that FlexDraw is NP-complete even if the inflexible edges form a matching. In fact, we show the stronger result of NP-hardness of instances with $O(n^\varepsilon)$ inflexible edges (for $\varepsilon > 0$) even if these edges are distributed evenly over the graph, i.e., they have pairwise distance $\Omega(n^{1-\varepsilon})$. This for example shows NP-hardness for instances with $O(\sqrt{n})$ inflexible edges with pairwise distances of $\Omega(\sqrt{n})$.

We adapt the proof of NP-hardness by Garg and Tamassia [GT01] for the case that all edges of an instance of FlexDraw are inflexible. For a given instance of Nae-3Sat (Not All Equal 3SAT) they show how to construct a graph G that admits an orthogonal representation without bends if and only if the instance of Nae-3Sat is satisfiable. The graph G is obtained by first constructing a graph F that has a unique planar embedding [GT01, Lemma 5.1] and replacing the edges of F by special st-graphs, the so called tendrils and wiggles. Both, tendrils and wiggles, have degree 1 at both poles and a unique planar embedding up to possibly a flip. It follows for each vertex v of G, that the cyclic order of incident edges around v is fixed up to a flip. This implies the following lemma.

Lemma 2.1 (Garg & Tamassia [GT01])**.** *FlexDraw is NP-hard, even if the order of edges around each vertex is fixed up to reversal.*

We assume that our instances do not contain degree-2 vertices; their incident edges can be replaced by a single edge with higher flexibility. In the following, we first show how to replace vertices of degree 3 by graphs of constant size such that each inflexible edge is incident to two vertices of degree 4. Afterwards, we can replace degree-4 vertices by smaller subgraphs with positive flexibility, which increases the distance between the inflexible edges. We start with the description of an st-graph that has either 1 or 2 bends in every valid orthogonal representation.

The wheel W_4 of size 4 consists of a 4-cycles $v_1, \dots, v_4$ together with a center u connected to each of the vertices $v_1, \dots, v_4$; see Figure 2.2a. We add the two vertices s and t together with the inflexible edges sv_1 and tv_2 to W_4. Moreover, we set the flexibility of v_3v_4 to 2 and the flexibilities of all other edges to 1. We call the resulting

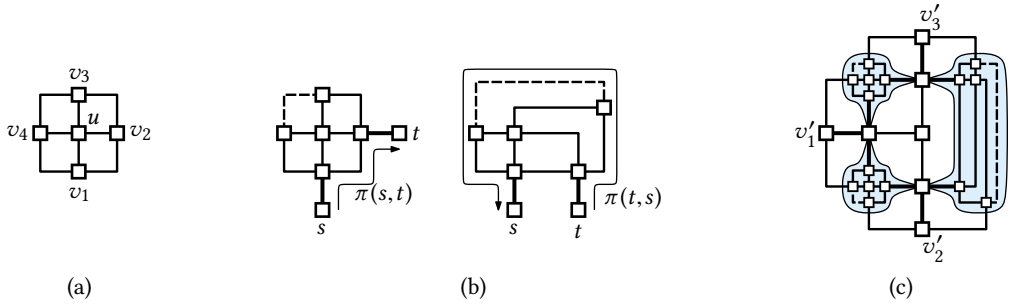


Figure 2.2: The bold edges are inflexible; dashed edges have flexibility 2; all other edges have flexibility 1. (a) The wheel W_4. (b) The bend gadget $B_{1,2}$. (c) The gadget W'_3 for replacing degree-3 vertices. The marked subgraphs are bend gadgets.

st-graph *bend gadget* and denote it by $B_{1,2}$. We only consider embeddings of $B_{1,2}$ where all vertices except for u lie on the outer face. Figure 2.2b shows two valid orthogonal representations of $B_{1,2}$, one with 1, the other with 2 bends. Clearly, the number of bends cannot be reduced to 0 (or increased above 2) without violating the flexibility constraints of edges on the path $\pi(s,t)$ (or on the path $\pi(t,s)$). Thus, $B_{1,2}$ has either 1 or 2 bends in every orthogonal representation. Moreover, if its embedding is fixed, then the direction of the bends is also fixed.

We now use the bend gadget as building block for a larger gadget. We start with the wheel W_3 of size 3 consisting of a triangle v_1, v_2, v_3 together with a center u connected to v_1, v_2, and v_3. The flexibilities of the edges incident to the center are set to 1, each edge in the triangle is replaced by a bend gadget $B_{1,2}$. To fix the embedding of the bend gadgets, we add three vertices v'_1, v'_2, and v'_3 connected with inflexible edges to v_1, v_2, and v_3, respectively, and connect them to the free incidences in the bend gadgets, as shown in Figure 2.2c. We denote the resulting graph by W'_3. Clearly, in the cycle of bend gadgets, two of them have one bend and the other has two bends in every valid orthogonal representation of W'_3. Thus, replacing a vertex v with incident edges e_1, e_2, and e_3 by W'_3, attaching the edge e_i to v'_i, yields an equivalent instance of FlexDraw. Note that such a replacement increases the degree of one incidence of e_1, e_2, and e_3 form 3 to 4. Moreover, every inflexible edge contained in W'_3 is incident to two vertices of degree 4. We obtain the following lemma.

Lemma 2.2. *FlexDraw is NP-hard, even if the endpoints of each inflexible edge have degree 4 and if the order of edges around each vertex is fixed up to reversal.*

Proof. Let G be an instance of FlexDraw such that the order of edges around each vertex is fixed up to reversal. As FlexDraw restricted to these kinds of instances is NP-hard, due to Lemma 2.1, it suffices to find an equivalent instance where additionally the endpoints of each inflexible have degree 4. Pairs of edges incident to a vertex of degree 2 can be simply replaced by an edge with higher flexibility. Thus, we can

assume that every vertex in G has degree 3 or degree 4. Replacing every degree-3 vertex incident to an inflexible edge by the subgraph W_3' described above clearly leads to an equivalent instance with the desired properties. □

Similar to the replacement of degree-3 vertices by W_3', we can replace degree-4 vertices by the wheel W_4, setting the flexibility of every edge of W_4 to 1. It is easy to see, that every valid orthogonal representation of W_4 has the same outer shape, i.e., a rectangle, with one of the vertices $v_1, \dots, v_4$ on each side; see Figure 2.2a. Thus, replacing a vertex v with incident edges $e_1, \dots, e_4$ (in this order) by W_4, attaching $e_1, \dots, e_4$ to the vertices $v_1, \dots, v_4$ yields an equivalent instance of FlexDraw. We obtain the following theorem.

Theorem 2.1. *FlexDraw is NP-complete even for instances of size n with $O(n^\varepsilon)$ inflexible edges with pairwise distance $\Omega(n^{1-\varepsilon})$.*

Proof. As FlexDraw is clearly in NP, it remains to show NP-hardness. Let G be the instance of FlexDraw such that the endpoints of each inflexible edge have degree 4 and such that the order of edges around each edge is fixed up to reversal. FlexDraw restricted to these kinds of instances is NP-hard due to Lemma 2.2. We show how to build an equivalent instance with $O(n^\varepsilon)$ inflexible edges with pairwise distance $\Omega(n^{1-\varepsilon})$ for any $\varepsilon > 0$.

Let e be an inflexible edge in G with incident vertices u and v, which both have degree 4. Replacing each of the vertices u and v by the wheel W_4 yields an equivalent instance of FlexDraw and the distance of e to every other inflexible edge is increased by a constant. Note that this does not increase the number of inflexible edges. Let n_G be the number of vertices in G. Applying this replacement $n_G^{1/\varepsilon-1}$ times to the vertices incident to each inflexible edge yields an equivalent instance G'. In G' every pair of inflexible edges has distance $\Omega(n_G^{1/\varepsilon-1})$. Moreover, G' has size $O(n_G^{1/\varepsilon})$, as we have n_G inflexible edges. Substituting $n_G^{1/\varepsilon}$ by n shows that we get an instance of size n with $O(n^\varepsilon)$ inflexible edges with pairwise distance $\Omega(n^{1-\varepsilon})$. □

Note that the instances described above may contain edges with flexibility larger than 1. We can get rid of that as follows. An edge e with flexibility $\text{flex}(e) > 0$ can have the same numbers of bends like the st-graph consisting of the wheel W_4 (Figure 2.2a) with the additional edges sv_1 with $\text{flex}(sv_1) = 1$ and tv_3 with $\text{flex}(tv_3) = \text{flex}(e) - 1$. Thus, we can successively replace edges with rotation above 1 by these kinds of subgraphs, leading to an equivalent instance where all edges have flexibility 1 or 0.

2.3 The General Algorithm

In this section we describe a general algorithm that can be used to solve OptimalFlexDraw by solving smaller subproblems for the different types of graph compositions

(series, parallel, and rigid; see Section 1.4.3). To this end, we start with the definition of cost functions for subgraphs, which is straightforward. The *cost function* $\text{cost}(\cdot)$ of an st-graph G is defined such that $\text{cost}(\beta)$ is the minimum cost of all orthogonal representations of G with β bends. The (σ,τ)-*cost function* $\text{cost}_\tau^\sigma(\cdot)$ of G is defined analogously by setting $\text{cost}_\tau^\sigma(\beta)$ to the minimum cost of all (σ,τ)-orthogonal representations of G with β bends. Clearly, $\sigma,\tau \in \{1,\dots 4\}$, though, for a fixed graph G, not all values may be possible. If for example $\deg(s) = 1$, then σ is 1 for every orthogonal representation of G. Note that there is a lower bound on the number of bends depending on σ and τ. For example, a $(2,2)$-orthogonal representation has at least one bend and thus $\text{cost}_2^2(0)$ is undefined. We formally set undefined values to ∞.

With the *cost functions* of G we refer to the collection of (σ,τ)-cost functions of G for all possible combinations of σ and τ. Let G be the composition of two or more (for a rigid composition) graphs $G_1,\dots,G_\ell$. Computing the cost functions of G assuming that the cost functions of $G_1,\dots,G_\ell$ are known is called *computing cost functions of a composition*. The following theorem states that the ability to compute cost functions of compositions suffices to solve OptimalFlexDraw. The terms T_S, T_P and $T_R(\ell)$ denote the time necessary to compute the cost functions of a series, a parallel, and a rigid composition with skeleton of size ℓ, respectively.

Theorem 2.2. *Let G be an st-graph containing the edge st. An optimal (σ,τ)-orthogonal representation of G with st on the outer face can be computed in $O(nT_S + nT_P + T_R(n))$ time.*

Proof. Let $\mathcal{T}$ be the SPQR-tree of G. To compute an optimal orthogonal representation of G with st on the outer face, we root $\mathcal{T}$ at the Q-node corresponding to st and traverse it bottom up. When processing a node μ, we compute the cost functions of $\text{pert}(\mu)$, which finally (in the root) yields the cost functions of the st-graph G and thus optimal (σ,τ)-orthogonal representations (for all possible values of σ and τ) with st on the outer face.

If μ is a **Q-node** but not the root, then $\text{pert}(\mu)$ is an edge and the cost function of this edge is given with the input.

If μ is an **S-node**, its pertinent graph can be obtained by applying multiple series compositions. Since the skeleton of an S-node leaves no embedding choice, we can compute the cost function of $\text{pert}(\mu)$ by successively computing the cost functions of the compositions, which takes $O(|\text{skel}(\mu)| \cdot T_S)$ time.

If μ is a **P-node**, then $\text{pert}(\mu)$ can be obtained by applying multiple parallel compositions. In contrast to S-nodes the skeleton of a P-node leaves an embedding choice, namely changing the order of the parallel edges. As composing the pertinent graphs of the children of μ in a specific order restricts the embedding of $\text{skel}(\mu)$, we cannot apply the compositions in an arbitrary order if $\text{skel}(\mu)$ contains more than two parallel edges (not counting the parent edge). However, since $\text{skel}(\mu)$ contains at most three parallel edges (due to the restriction to degree 4), we can try all composition orders

and take the minimum over the resulting cost functions. As there are only constantly many orders and for each order a constant number of compositions is performed, computing the cost function of $\text{pert}(\mu)$ takes $O(T_P)$ time.

If μ is an **R-node**, the pertinent graph of μ is the rigid composition of the pertinent graphs of its children with respect to the skeleton $\text{skel}(\mu)$. Thus, the cost functions of $\text{pert}(\mu)$ can be computed in $O(T_R(|\text{skel}(\mu)|))$ time.

If μ is the **root**, i.e., the Q-node corresponding to st, then $\text{pert}(\mu) = G$ is a parallel composition of the pertinent graph of the child of μ and the edge st and thus its cost function can be computed in $O(T_P)$ time.

As the total size of S-node skeletons, the number of P-nodes and the total size of R-node skeletons is linear in the size of G, the running time is in $O(n \cdot T_S + n \cdot T_P + T_R(n))$. □

Applying Theorem 2.2 for each pair of adjacent nodes as poles in a given instance of OptimalFlexDraw yields the following corollary.

Corollary 2.1. *OptimalFlexDraw can be solved in $O(n \cdot (nT_S + nT_P + T_R(n)))$ time for biconnected graphs.*

In the following, we extend this result to the case where G may contain cutvertices. The extension is straightforward, however, there is one pitfall. Given two blocks B_1 and B_2 sharing a cutvertex v such that v has degree 2 in B_1 and B_2, we have to ensure for both blocks that v does not form an angle of 180°. Thus, for a given graph G, we get for each block a list of vertices and we restrict the set of all orthogonal representations of G to those where these vertices form 90° angles. We call these orthogonal representations *restricted orthogonal representations*. Moreover, we call the resulting cost functions *restricted cost functions*. We use the terms T_S^r, T_P^r and $T_R^r(\ell)$ to denote the time necessary to compute the *restricted* cost functions of a series, a parallel, and a rigid composition, respectively. We get the following extension of the previous results.

Theorem 2.3. *OptimalFlexDraw can be solved in $O(n \cdot (nT_S^r + nT_P^r + T_R^r(n)))$ time.*

Proof. Let G be an instance of OptimalFlexDraw. We use the BC-tree (Block–Cutvertex Tree) of G to represent all possible ways of combining embeddings of the blocks of G to an embedding of G. The *BC-tree* $\mathcal{T}$ of G contains a B-node for each block of G, a C-node for each cutvertex of G and an edge between a C-node and a B-node if and only if the corresponding cutvertex is contained in the corresponding block, respectively.

Rooting $\mathcal{T}$ at some B-node restricts the embeddings of the blocks as follows. Let μ be a B-node (but not the root) corresponding to a block B and let v be the cutvertex corresponding to the parent of μ. Then the embedding of B is required to have v on its outer face. It is easy to see that every embedding of G is such a restricted embedding

with respect to some root of $\mathcal{T}$. Thus, it suffices to consider each B-node of $\mathcal{T}$ as root and restrict the embeddings as described above.

Before we deal with the BC-tree $\mathcal{T}$, we preprocess each block B of G. Let v be a cutvertex of B. For an edge e incident to v, we can use Theorem 2.2 to compute an optimal orthogonal representation of B with e on the outer face in $O(n \cdot T_S + n \cdot T_P + T_R(n))$ time. Since ever orthogonal representation with v on the outer face has one of its incident edges on the outer face, we can simply force each of these edges to the outer face once, to get an optimal orthogonal representation of B with v on the outer face. Clearly, using the computation of restricted cost functions yields an optimal restricted orthogonal representation. Doing this for each block of G and for each cutvertex in this block leads to a total running time of $O(n \cdot (n \cdot T_S + n \cdot T_P + T_R(n)))$. Moreover, we can compute an optimal restricted orthogonal representation of each block (without forcing a vertex to the outer face) with the same running time (Corollary 2.1).

To compute an optimal orthogonal representation of G we choose every B-node of the BC-tree $\mathcal{T}$ as root and consider for the block corresponding to the root the optimal orthogonal representation (without forcing vertices to the outer face). For all other blocks we consider the optimal orthogonal representation with the cutvertex corresponding to its parent on the outer face. Note that these orthogonal representations can be easily combined to an orthogonal representation of the whole graph, as we enforce angles of 90° at vertices of degree 2, if they have degree 2 in another block. The minimum over all roots leads to an optimal orthogonal representation. As computing this minimum takes $O(n^2)$ time, it is dominated by the running time necessary to compute the orthogonal representation of the blocks. □

Note that Theorem 2.3 provides a framework for uniform treatment of bend minimization over all planar embeddings in orthogonal drawings. In particular, the polynomial-time algorithm for FlexDraw with positive flexibility [Blä+14] can be expressed in this way. There, all resulting cost functions of st-graphs are 0 on a non-empty interval containing 0 (with one minor exception) and ∞, otherwise. Thus, the cost functions of the compositions can be computed using Tamassia's flow network. The results on OptimalFlexDraw [BRW13] can be expressed similarly. When restricting the number of bends of each st-graph occurring in the composition to 3, all resulting cost functions are convex (with one minor exception). Thus, Tamassia's flow network can again be used to compute the cost functions of the compositions. The overall optimality follows from the fact that there exists an optimal solution that can be composed in such a way. In the following sections we see two further applications of this framework, resulting in efficient algorithms.

2.4 Series-Parallel Graphs

In this section we show that the cost functions of a series composition (Lemma 2.3) and a parallel composition (Lemma 2.4) can be computed efficiently. Using our framework, this leads to a polynomial-time algorithm for OptimalFlexDraw for series-parallel graphs with monotone cost functions (Theorem 2.4). We note that this is only a slight extension to the results by Di Battista et al. [DLV98]. However, it shows the easy applicability of the above framework before diving into the more complicated FPT-algorithm in the following section.

Lemma 2.3. *If the (restricted) cost functions of two st-graphs are ∞ for bend numbers larger than ℓ, the (restricted) cost functions of their series composition can be computed in $O(\ell^2)$ time.*

Proof. We first consider the case of non-restricted cost functions. Let G_1 and G_2 be the two st-graphs with poles s_1, t_1 and s_2, t_2, respectively, and let G be their series composition with poles $s = s_1$ and $t = t_2$. For each of the constantly many valid combinations of σ and τ, we compute the (σ,τ)-cost function separately. Assume for the following, that σ and τ are fixed. Since G_1 and G_2 both have at most ℓ bends, G can only have $O(\ell)$ possible values for the number of bends β. We fix the value β and show how to compute $\mathrm{cost}_\tau^\sigma(\beta)$ in $O(\ell)$ time.

Let $\mathcal{R}$ be a (σ,τ)-orthogonal representation with β bends and let $\mathcal{R}_1$ and $\mathcal{R}_2$ be the (σ_1,τ_1)- and (σ_2,τ_2)-orthogonal representations induced for G_1 and G_2, respectively. Obviously, $\sigma_1 = \sigma$ and $\tau_2 = \tau$ holds. However, there are the following other parameters that may vary (although they may restrict each other). The parameters τ_1 and σ_2; the number of bends β_1 and β_2 of $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively; the possibility that for $i \in \{1,2\}$ the number of bends of $\mathcal{R}_i$ are determined by $\pi(s_i,t_i)$ or by $\pi(t_i,s_i)$, i.e., $\beta_i = -\mathrm{rot}(\pi(s_i,t_i))$ or $\beta_i = -\mathrm{rot}(\pi(t_i,s_i))$; and finally, the rotations at the vertex v in the outer face, where v is the vertex of G belonging to both, G_1 and G_2.

Assume we fixed the parameters τ_1 and σ_2, the choice by which paths β_1 and β_2 are determined, the rotations at the vertex v, and the number of bends β_1 of $\mathcal{R}_1$. Then there is no choice left for the number of bends β_2 of $\mathcal{R}_2$, as choosing a different value for β_2 also changes the number of bends β of G, which was assumed to be fixed. As each of the parameters can have only a constant number of values except for β_1, which can have $O(\ell)$ different values, there are only $O(\ell)$ possible choices in total. For each of these choices, we get a (σ,τ)-orthogonal representation of G with β bends and cost $\mathrm{cost}_{\tau_1}^{\sigma_1}(\beta_1) + \mathrm{cost}_{\tau_2}^{\sigma_2}(\beta_2)$. By taking the minimum cost over all these choices we get the desired value $\mathrm{cost}_\tau^\sigma(\beta)$ in $O(\ell)$ time.

If we consider restricted cost functions, it may happen that the vertex v has degree 2. Then we need to enforce an angle of 90° there. Obviously, this constraint can be easily added to the described algorithm. □

Lemma 2.4. *If the (restricted) cost functions of two st-graphs are ∞ for bend numbers larger than ℓ, the (restricted) cost functions of their parallel composition can be computed in $O(\ell)$ time.*

Proof. If the composition G of G_1 and G_2 has β bends, either the graph G_1 or the graph G_2 also has β bends. Thus, the cost function of G is ∞ for bend numbers larger than ℓ. Let the number of bends of G be fixed to β. Similar to the proof of Lemma 2.3, there are the following parameters. The number of bends β_1 and β_2 of G_1 and G_2; σ_i and τ_i for $i \in \{1,2\}$; σ and τ; the order of the two graphs; and the decision whether $\pi(s,t)$ or $\pi(t,s)$ determines the number of bends of G. All parameters except for β_1 and β_2 have $O(1)$ possible values. As mentioned before, we have $\beta = \beta_1$ or $\beta = \beta_2$. In the former case, fixing all parameters except for β_2 leaves no choice for β_2. The case of $\beta = \beta_2$ leaves no choice for β_1. Thus, each of the $O(\ell)$ values can be computed in $O(1)$ time, which concludes the proof. □

Theorem 2.4. *For series-parallel graphs with monotone cost functions* OptimalFlexDraw *can be solved in $O(n^4)$ time.*

Proof. To solve OptimalFlexDraw, we use Theorem 2.3. As the graphs we consider here are series parallel, it suffices to give algorithms that compute the cost functions of series and parallel compositions. Applying Lemma 2.3 and Lemma 2.4 gives us running times $T_S \in O(\ell^2)$ and $T_P \in O(\ell)$ for these compositions. In the following, we show that it suffices to compute the cost functions for a linear number of bends, leading to running times $T_s \in O(n^2)$ and $T_P \in O(n)$. Together with the time stated by Theorem 2.3, this gives us a total running time of $O(n^4)$.

Let G be an st-graph with monotone cost functions assigned to the edges. We show the existence of an optimal orthogonal representation of G such that every split component of G has $O(n)$ bends. To this end, consider the flow network N introduced by Tamassia [Tam87] and let d be the total demand of all its sinks. Let $\mathcal{R}$ be an optimal orthogonal representation of G such that a split component H has at least $d+1$ bends. Then one of the two faces incident to edges of H and to edges of $G - H$ has at least $d+1$ units of outgoing flow. As the total demand of sinks in the flow network is only d, there must exist a directed cycle C in N such that the flow on each of the arcs in C is at least 1. Reducing the flow on C by 1 yields a new orthogonal representation and as the number of bends on no edge is increased, the cost does not increase. As in every step the total amount of flow is decreased, the process stops after finitely many steps. The result is an optimal orthogonal representation of G such that each split component has at most d bends. Thus, we can restrict our search to orthogonal representations in which each split component has only up to d bends. This can be done by implicitly setting the costs to ∞ for larger values than d. This concludes the proof, as $d \in O(n)$ holds. □

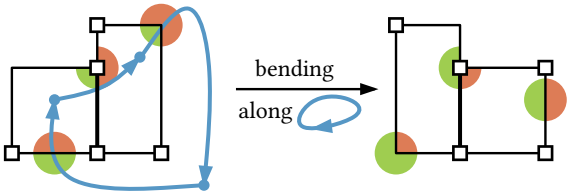


Figure 2.3: An orthogonal representation (the bold edge is inflexible, all other edges have flexibility 1). Bending along the valid cycle (blue) increases the green and decreases the red angles.

2.5 An FPT-Algorithm for General Graphs

Let G be an instance of FlexDraw. We call an edge in G *critical* if it is inflexible and at least one of its endpoints has degree 4. We call the instance G of FlexDraw k-critical, if it contains exactly k critical edges. An inflexible edge that is not critical is *semi-critical*. The poles s and t of an st-graph G are considered to have additional neighbors (which comes from the fact that we usually consider st-graphs to be subgraphs of larger graphs). More precisely, inflexible edges incident to the pole s (or t) are already *critical* if $\deg(s) \geq 2$ (or $\deg(t) \geq 2$). In the following, we first study cost functions of k-critical st-graphs. Afterwards, we show how to use the insights we got to give an FPT-algorithm for k-critical instances of FlexDraw.

2.5.1 The Cost Functions of k-Critical Instances

Let G be an st-graph and let $\mathcal{R}$ be a valid orthogonal representation of G. We define an operation that transforms $\mathcal{R}$ into another valid orthogonal representation of G. Let $G^\star$ be the *double directed* dual graph of G, i.e., each edge e of G with incident faces g and f corresponds to the two dual edges (g,f) and (f,g). We call a dual edge $e^\star = (g,f)$ of e *valid* if one of the following conditions holds.

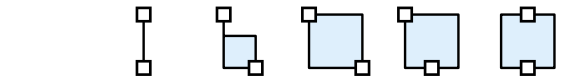
(I) $\text{rot}(e_f) < \text{flex}(e)$ (which is equivalent to $-\text{rot}(e_g) < \text{flex}(e)$).

(II) $\text{rot}(v_f) < 1$ where v is an endpoint of e but not a pole.

A simple directed cycle $C^\star$ in $G^\star$ consisting of valid edges is called *valid cycle*. Then *bending along* $C^\star$ changes the orthogonal representation $\mathcal{R}$ as follows; see Figure 2.3. Let $e^\star = (g,f)$ be an edge in $C^\star$ with primal edge e. If $e^\star$ is valid due to Condition (I), we reduce $\text{rot}(e_g)$ by 1 and increase $\text{rot}(e_f)$ by 1. Otherwise, if Condition (II) holds, we reduce $\text{rot}(v_g)$ by 1 and increase $\text{rot}(v_f)$ by 1, where v is the vertex incident to e with $\text{rot}(v_f) < 1$.

Lemma 2.5. *Let G be an st-graph with a valid (σ,τ)-orthogonal representation $\mathcal{R}$. Bending along a valid cycle $C^\star$ yields a valid (σ,τ)-orthogonal representation.*

Proof. First, we show that the resulting rotations still describe an orthogonal representation. Afterwards, we show that this orthogonal representation is also valid and that



(σ,τ) =	$(1,1)$	$(1,2)$	$(2,2)$	$(2,3)$	$(3,3)$
β_{low} =	0	1	1	2	2

Figure 2.4: Illustration of Fact 2.1 for some values of σ and τ.

it is a (σ,τ)-orthogonal representation. Let $e^\star = (g,f)$ be an edge in $C^\star$ with primal edge e. If Condition (I) holds, then $\text{rot}(e_g)$ is reduced by 1 and $\text{rot}(e_f)$ is increased by 1 and thus $\text{rot}(e_g) = -\text{rot}(e_f)$ remains true. Otherwise, Condition (II) holds and thus $\text{rot}(v_g)$ is reduced by 1 and $\text{rot}(v_f)$ is increased by 1. The total rotation around v does obviously not change. Moreover, both rotations remain in the interval $[-1,1]$. Finally, the incoming arc to a face f in $C^\star$ increases the rotation around f by 1 and the outgoing arc decreases it by 1. Thus, the total rotation around each face remains as it was.

It remains to show that the resulting orthogonal representation is a valid (σ,τ)-orthogonal representation. First, Condition (I) ensures that we never increase the number of bends of an edge e above $\text{flex}(e)$. Moreover, due to the exception in Condition (II) where v is one of the poles, we never change the rotation of one of the poles. Thus the number of free incidences to the outer face are not changed. □

As mentioned in Section 2.3, depending on σ and τ, there is a lower bound on the number of bends of (σ,τ)-orthogonal representations. We denote this lower bound by β_{low}; see Figure 2.4.

Fact 2.1. *A (σ,τ)-orthogonal representation has at least $\beta_{\text{low}} = \left\lceil \frac{\sigma+\tau}{2} \right\rceil - 1$ bends.*

For a valid orthogonal representation with a large number of bends, the following lemma states that we can reduce its bends by bending along a valid cycle. This can later be used to show that the cost function of an st-graph is 0 on a significantly large interval. Or in other words, arbitrary alterations of cost 0 and cost ∞ that are hard to handle only occur on a small interval (depending on k). The lemma and its proof are a generalization of Lemma 1 from [Blä+14] that incorporates inflexible edges. For $\sigma = \tau = 3$ a slightly weaker result holds.

Lemma 2.6. *Let G be a k-critical st-graph and let $\mathcal{R}$ be a valid (σ,τ)-orthogonal representation with $\sigma+\tau \leq 5$. If $-\text{rot}(\pi(t,s)) \geq \beta_{\text{low}} + k + 1$ holds, then there exists a valid cycle $C^\star$ such that bending $\mathcal{R}$ along $C^\star$ reduces $-\text{rot}(\pi(t,s))$ by 1.*

Proof. We show the existence of a valid cycle $C^\star$ such that s and t lie to the left and right of $C^\star$, respectively. Obviously, such a cycle must contain the outer face. The edge in $C^\star$ having the outer face as target ensures that the rotation of an edge or a vertex of $\pi(t,s)$ is increased by 1 (which is the same as reducing $-\text{rot}(\pi(t,s))$ by 1), where this vertex is neither s nor t (due to the exception of Condition (II)). Thus, $\text{rot}(\pi(t,s))$

is increased by 1 when bending along $C^\star$ and thus $C^\star$ is the desired cycle. We first show the following claim.

Claim 1. *There exists a valid edge $e^\star$ that either has the outer face as source and corresponds to a primal edge e on the path $\pi(s,t)$, or is a loop with s to its left and t to right.*

Assume the claimed edge $e^\star$ does not exist. We first show that the following inequality follows from this assumption. Afterwards, we show that this leads to a contradiction to the inequality in the statement of the lemma.

$$\operatorname{rot}(\pi(s,t)) \leq \begin{cases} k, & \text{if } \deg(s) = \deg(t) = 1 \\ k-1, & \text{otherwise} \end{cases} \tag{2.1}$$

We first show this inequality for the case where we have **no critical and no semi-critical edges, in particular $k = 0$**. We consider the rotation of edges and vertices on $\pi(s,t)$ in the outer face g. If an edge or vertex has two incidences to g, we implicitly consider the incidence corresponding to $\pi(s,t)$. Recall that the rotation along $\pi(s,t)$ is the sum over the rotations of its edges and of its internal vertices. The rotation of every edge e is $\operatorname{rot}(e_g) = -\operatorname{flex}(e)$ as otherwise $e^\star = (g,f)$ would be a valid edge due to Condition (I). At an internal vertex v we obviously have $\operatorname{rot}(v_g) \leq 1$, as larger rotations are not possible at vertices. Hence, as the flexibility of every edge is at least 1 and we have an internal vertex less than we have edges, we get $\operatorname{rot}(\pi(s,t)) \leq -1$ and thus Equation (2.1) is satisfied.

Next, we allow **semi-critical edges, but no critical edges ($k = 0$ remains)**. If $\pi(s,t)$ contains a semi-critical edge, it has a rotation of 0 (instead of -1 for normal edges). Note that we still assume that there is no critical edge in $\pi(s,t)$, i.e., $k = 0$. Moreover, if an internal vertex v is incident to a semi-critical edge, it cannot have degree 4. In this case, there must be a face incident to v such that v has rotation at most 0 in this face. If this face was not g, Condition (II) would be satisfied. Thus, $\operatorname{rot}(v_g) \leq 0$ follows for this case. Consider the decomposition of $\pi(s,t)$ into maximal subpaths consisting of semi-critical and normal edges. If follows that each subpath consisting of semi-critical and normal edges hat rotation at most 0 and -1, respectively. Moreover, the rotation at vertices between two subpaths is 0. Hence, if $\pi(s,t)$ contains at least one edge that is not semi-critical, we again get $\operatorname{rot}(\pi(s,t)) \leq -1$ and thus Equation (2.1) is satisfied. On the other hand, if $\pi(s,t)$ consists of semi-critical edges, we get the weaker inequality $\operatorname{rot}(\pi(s,t)) \leq 0$. If $\deg(s) = \deg(t) = 1$ holds, Equation (2.1) is still satisfied as we have to show a weaker inequality in this case. Otherwise, one of the poles has degree at least 2 and thus the edges incident to it cannot be semi-critical by definition. Thus, the path $\pi(s,t)$ cannot consist of semi-critical edges.

Finally, we allow **critical edges, i.e., $k \geq 0$**. If $\pi(s,t)$ contains critical edges, we first consider these edges to have flexibility 1, leading to Equation (2.1) with $k = 0$.

Replacing an edge with flexibility 1 by an edge with flexibility 0 increases the rotation along $\pi(s,t)$ by at most 1. As $\pi(s,t)$ contains at most k critical edges, $\text{rot}(\pi(s,t))$ is increased by at most k yielding Equation (2.1).

In the case that $\deg(s) = \deg(t) = 1$, the equation $\text{rot}(\pi(s,t)) = -\text{rot}(\pi(t,s))$ holds. Equation (2.1) together with the inequality in the statement of the lemma leads to $k \geq \beta_{\text{low}} + k + 1$, which is a contradiction. In the following, we only consider the case where $\deg(s) = \deg(t) = 1$ does not hold. Since the total rotation around the outer face sums up to -4, we get the following equation.

$$\text{rot}(\pi(s,t)) + \text{rot}(\pi(t,s)) + \text{rot}(s_g) + \text{rot}(t_g) = -4$$

Recall that $\text{rot}(s_g) = \sigma - 3$ and $\text{rot}(t_g) = \tau - 3$. Using Equation (2.1) ($\deg(s) = \deg(t) = 1$ does not hold) and the inequality given in the lemmas precondition, we obtain the following.

$$\begin{aligned} & \Big(k-1\Big) - \Big(\overbrace{\Big\lceil \frac{\sigma+\tau}{2} \Big\rceil - 1}^{\beta_{\text{low}}} + k + 1\Big) + \Big(\sigma - 3\Big) + \Big(\tau - 3\Big) \geq -4 \\ \Leftrightarrow \quad & -\Big\lceil \frac{\sigma+\tau}{2} \Big\rceil + (\sigma+\tau) \geq 3 \\ \Leftrightarrow \quad & \Big\lfloor \frac{\sigma+\tau}{2} \Big\rfloor \geq 3 \end{aligned} \tag{2.2}$$

Recall that $\sigma + \tau \leq 5$ is a requirement of the lemma. Thus, Equation (2.2) is a contradiction, which concludes the proof of Claim 1.

Claim 2. *The valid cycle $C^\star$ exists.*

Let $e^\star$ be the valid edge existing due to Claim 1. If $e^\star$ is a loop with s to its left and t to its right, then $C^\star = e^\star$ is the desired valid cycle. This case will serve as base case for a structural induction.

Let $e^\star = (g, f)$ be a valid edge dual to e having the outer face g as source. As $e^\star$ is not a loop, the graph $G - e$ is still connected and thus s and t are contained in the same block of the graph $G - e + st$. Let H be this block (without st) and let $\mathcal{S}$ be the orthogonal representation of H induced by $\mathcal{R}$. Then H is a k-critical st-graph, as H is a subgraph of G and $H + st$ is biconnected. Moreover, the path $\pi(t,s)$ is completely contained in H and thus its rotation does not change. Hence, all conditions for Lemma 2.6 are satisfied and since H contains fewer edges than G, we know by induction that there exists a valid cycle $C_H^\star$ such that bending $\mathcal{S}$ along $C_H^\star$ reduces $-\text{rot}(\pi(t,s))$ by 1. As the dual graph $H^\star$ of H can be obtained from $G^\star$ by first contracting $e^\star$ and then taking a subgraph, all edges contained in $H^\star$ were already contained in $G^\star$. Moreover, all valid edges in $H^\star$ are also valid in $G^\star$ and thus each edge in $C_H^\star$ corresponds to a valid edge in $G^\star$. If these valid edges form a cycle in $G^\star$, then this is the desired cycle

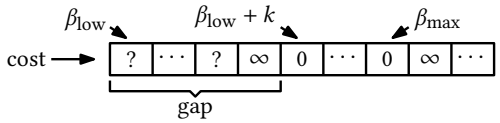


Figure 2.5: A cost function with gap k.

$C^\star$. Otherwise, one of the two edges in $C^\star_H$ incident to the outer face of H is in $G^\star$ incident to the outer face g of G and the other is incident to the face f of G. In this case the edges of $C^\star_H$ from in $G^\star$ a path from f to g and thus adding the edge $e^\star$ yields the cycle $C^\star$, which concludes the proof of Claim 2 and thus of this lemma. □

We get the following slightly weaker result for the case $\sigma = \tau = 3$.

Lemma 2.7. *Let G be a k-critical st-graph and let $\mathcal{R}$ be a valid $(3,3)$-orthogonal representation. If $-\operatorname{rot}(\pi(t,s)) \geq \beta_{\text{low}} + k + 2$ holds, then there exists a valid cycle $C^\star$ such that bending $\mathcal{R}$ along $C^\star$ reduces $-\operatorname{rot}(\pi(t,s))$ by 1.*

Proof. Since $\sigma = \tau = 3$ holds, we have $\beta_{\text{low}} = 2$ and thus $-\operatorname{rot}(\pi(t,s)) \geq k + 4$. We add an edge $e = ss'$ with flexibility 1 to G, where s' is a new vertex, and consider the orthogonal representation $\mathcal{R}'$ of $G + e$ where e has one bend such that e contributes a rotation of 1 to $\pi(t,s')$. Since the rotation at s in the outer face is 1, we have $\operatorname{rot}(\pi(t,s')) = \operatorname{rot}(\pi(t,s)) + 2$. If follows that $-\operatorname{rot}(\pi(t,s')) \geq k + 4 - 2 = k + 2$ holds. Since $\mathcal{R}'$ is a $(1,3)$ orthogonal representation of $G + e$, and since the lower bound β'_{low} is 1 for $(1,3)$ orthogonal representations, the precondition of Lemma 2.6, namely the inequality $-\operatorname{rot}(\pi(t,s')) \geq \beta'_{\text{low}} + k + 1$, is satisfied, which concludes the proof. □

The previous lemmas basically show that the existence of a valid orthogonal representation with a lot of bends implies the existence of valid orthogonal representations for a "large" interval of bend numbers. This is made more precise in the following.

Let $\mathcal{B}^\sigma_\tau$ be the set containing an integer β if and only if G admits a valid (σ,τ)-orthogonal representation with β bends. Assume G admits a valid (σ,τ)-orthogonal representation, i.e., $\mathcal{B}^\sigma_\tau$ is not empty. We define the *maximum bend value* $\beta_{\max}$ to be the maximum in $\mathcal{B}^\sigma_\tau$. Moreover, let $\beta \in \mathcal{B}^\sigma_\tau$ be the smallest value, such that every integer between β and $\beta_{\max}$ is contained in $\mathcal{B}^\sigma_\tau$. Then we call the interval $[\beta_{\text{low}}, \beta - 1]$ the (σ,τ)-*gap* of G. The value $\beta - \beta_{\text{low}}$ is also called the (σ,τ)-*gap* of G; see Figure 2.5.

Lemma 2.8. *The (σ,τ)-gap of a k-critical st-graph G is at most k if $\sigma + \tau \leq 5$. The $(3,3)$ gap of G is at most $k + 1$.*

Proof. In the following, we assume $\sigma + \tau \leq 5$; the case $\sigma = \tau = 3$ works literally the same when replacing Lemma 2.6 by Lemma 2.7. Let $\mathcal{R}$ be a valid (σ,τ)-orthogonal representation with $\beta \geq \beta_{\text{low}} + k + 1$ bends. We show the existence of a valid (σ,τ)-orthogonal representation with $\beta - 1$ bends. It follows that the number of bends can be reduced step by step down to $\beta_{\text{low}} + k$, which shows that the gap is at most k.

As $\mathcal{R}$ has β bends, either $-\operatorname{rot}(\pi(s,t)) = \beta$ or $-\operatorname{rot}(\pi(t,s)) = \beta$. Without loss of generality, we assume $-\operatorname{rot}(\pi(t,s)) = \beta \geq \beta_{\text{low}} + k + 1$. Due to Lemma 2.6 there exists a valid cycle $C^\star$, such that bending along $C^\star$ reduces $-\operatorname{rot}(\pi(t,s))$ by 1. This also reduces the number of bends by 1 (and thus yields the desired orthogonal representation) if $-\operatorname{rot}(\pi(s,t))$ is not increased above $\beta - 1$. Assume for a contradiction that $-\operatorname{rot}(\pi(s,t))$ was increased above $\beta - 1$. Then in the resulting orthogonal representation $-\operatorname{rot}(\pi(s,t))$ is greater than β_{low} and $-\operatorname{rot}(\pi(t,s))$ is at least β_{low}. It follows, that every (σ,τ)-orthogonal representation has more than β_{low} bends, which contradicts the fact, that β_{low} is a tight lower bound. □

The following lemma basically expresses the gap of an st-graph in terms of the rotation along $\pi(s,t)$ instead of the number of bends.

Lemma 2.9. *Let G be an st-graph with (σ,τ)-gap k. The set $\{\rho \mid G$ admits a valid (σ,τ)-orthogonal representation with $\operatorname{rot}(\pi(s,t)) = \rho\}$ is the union of at most $k+1$ intervals.*

Proof. Recall that an orthogonal representation has β bends if either $-\operatorname{rot}(\pi(s,t)) = \beta$ or $-\operatorname{rot}(\pi(t,s)) = \beta$. We first consider the case that $-\operatorname{rot}(\pi(s,t)) = \beta$ for any number of bends $\beta \in [\beta_{\text{low}}, \beta_{\text{max}}]$.

By the definition of the gap, there exists a valid orthogonal representation for $-\operatorname{rot}(\pi(s,t)) \in [\beta_{\text{low}} + k, \beta_{\text{max}}]$, which forms the first interval. Moreover, G does not admit a valid orthogonal representation with $\beta_{\text{low}} + k - 1$ bends, since the gap would be smaller otherwise. Thus it remains to cover all allowed values contained in $[\beta_{\text{low}}, \beta_{\text{low}} + k - 2]$ by intervals. In the worst case, exactly every second value is possible. As $[\beta_{\text{low}}, \beta_{\text{low}} + k - 2]$ contains $k-1$ integers, this results in $\lceil (k-1)/2 \rceil$ intervals of size 1. Thus, we can cover all allowed values for $\operatorname{rot}(\pi(s,t))$ in case $-\operatorname{rot}(\pi(s,t)) \in [\beta_{\text{low}} + k, \beta_{\text{max}}]$ holds using only $\lceil (k-1)/2 \rceil + 1$ intervals.

It remains to consider the case where G has β bends due to the fact that the equation $-\operatorname{rot}(\pi(t,s)) = \beta$ holds. With the same argument we can cover all possible values of $\pi(t,s)$ using $\lceil (k-1)/2 \rceil + 1$ intervals. As $\operatorname{rot}(\pi(s,t))$ equals $-\operatorname{rot}(\pi(t,s))$ shifted by some constant, we can cover all allowed values for $\operatorname{rot}(\pi(s,t))$ using $2 \cdot \lceil (k-1)/2 \rceil + 2$ intervals. If $k-1$ is even, this evaluates to $k+1$ yielding the statement of the lemma. If $k-1$ is odd and we assume the above described worst case, then we need one additional interval. However, in this case there must exists a valid orthogonal representation with β_{low} bends and we counted two intervals for this bend number, namely for the case $-\operatorname{rot}(\pi(s,t)) = \beta_{\text{low}}$ and $-\operatorname{rot}(\pi(t,s)) = \beta_{\text{low}}$. We show that a single interval suffices to cover both cases by showing that either $-\operatorname{rot}(\pi(s,t)) = \beta_{\text{low}}$ or $-\operatorname{rot}(\pi(s,t)) = \beta_{\text{low}} - 1$ holds if $-\operatorname{rot}(\pi(t,s)) = \beta_{\text{low}}$. This again leads to the desired $k+1$ intervals.

As the rotation around the outer face is -4, the equation $-\operatorname{rot}(\pi(s,t)) = \sigma + \tau -$

$2 + \text{rot}(\pi(t,s))$ holds. For $-\text{rot}(\pi(t,s)) = \beta_{\text{low}}$ we get the following.

$$\sigma + \tau - 2 - \beta_{\text{low}} = \sigma + \tau - 2 - \left\lceil \frac{\sigma + \tau}{2} \right\rceil + 1 = \left\lfloor \frac{\sigma + \tau}{2} \right\rfloor - 1$$

If $\sigma + \tau$ is even, this is equal to β_{low}, otherwise it is equal to $\beta_{\text{low}} - 1$, which concludes the proof. □

2.5.2 Computing the Cost Functions of Compositions

Let G be a graph with fixed planar embedding. We describe a flow network, similar to the one by Tamassia [Tam87] that can be used to compute orthogonal representations of graphs with thick edges. In general, we consider a flow network to be a directed graph with a lower and an upper bound assigned to every edge and a demand assigned to every vertex. The bounds and demands can be negative. An assignment of flow-values to the edges is a feasible flow if it satisfies the following properties. The flow-value of each edge is at least its lower and at most its upper bound. For every vertex the flow on incoming edges minus the flow on outgoing edges must equal its demand.

We define the flow network N as follows. The network N contains a node for each vertex of G, the *vertex nodes*, each face of G, the *face nodes*, and each edge of G, the *edge nodes*. Moreover, N contains arcs from each vertex to all incident faces, the *vertex-face arcs*, and similarly from each edge to both incident faces, the *edge-face arcs*. We interpret an orthogonal representation $\mathcal{R}$ of G as a flow in N. A rotation $\text{rot}(e_f)$ of an edge e in the face f corresponds to the same amount of flow on the edge-face arc from e to f. Similarly, for a vertex v incident to f the rotation $\text{rot}(v_f)$ corresponds to the flow from v to f.

Obviously, the properties (1)–(4) of an orthogonal representation are satisfied if and only if the following conditions hold for the flow (note that we allow G to have thick edges).

(1) The total amount of flow on arcs incident to a face node is 4 (-4 for the outer face).

(2) The flow on the two arcs incident to an edge node stemming from a (σ,τ)-edge sums up to $2 - (\sigma + \tau)$.

(3) The total amount of flow on arcs incident to a vertex node, corresponding to the vertex v with incident edges $e_1,\dots,e_\ell$ occupying $\sigma_1,\dots,\sigma_\ell$ incidences of v is $\sum(\sigma_i + 1) - 4$.

(4) The flow on vertex-face arcs lies in the range $[-2,1]$.

Properties (1)–(3) are equivalent to the flow conservation requirement when setting appropriate demands. Moreover, property (4) is equivalent to the capacity constraints in a flow network when setting the lower and upper bounds of vertex-face arcs to -2

and 1, respectively. In the following, we use this flow network to compute the cost function of a rigid composition of graphs. The term $T_{\text{flow}}(\ell)$ denotes the time necessary to compute a maximal flow in a planar flow network of size ℓ.

Lemma 2.10. *The (restricted) cost functions of a rigid composition of ℓ graphs can be computed in $O(2^k \cdot T_{\text{flow}}(\ell))$ time if the resulting graph is k-critical.*

Proof. First note that in case of a rigid composition, computing "restricted" cost functions makes only a difference for the poles of the skeleton (as all other vertices have degree at least 3). However, enforcing 90° angles for the poles is already covered by the number of incidences the resulting graph occupies at its poles.

Let H be the skeleton of the rigid composition of the graphs $G_1, \ldots, G_\ell$ and let G be the resulting graph with poles s and t. Before we show how to compute orthogonal representations of G, we show that the number of incidences σ_i and τ_i a subgraph G_i occupies at its poles s_i and t_i is (almost) fixed. Assume s_i is not one of the poles s or t of G. Then s_i has at least three incident edges in the skeleton H as $H + st$ is triconnected. Thus, the subgraph G_i occupies at most two incidences in any orthogonal representation of G, and hence s_i has either degree 1 or degree 2 in G_i. In the former case σ_i is 1, in the latter σ_i has to be 2. If s_i is one of the poles of G, then it may happen that G_i occupies incidences in some orthogonal representations of G and three incidences in another orthogonal representation. However, this results in a constant number of combinations and thus we can assume that the values σ_i and τ_i are fixed for $i \in \{1, \ldots, \ell\}$.

To test whether G admits a valid (σ, τ)-orthogonal representation, we can instead check the existence of a valid orthogonal representation of H using thick edges for the graphs $G_1, \ldots, G_\ell$ (more precisely, we use a (σ_i, τ_i)-edge for G_i). To ensure that substituting the thick edges with the subgraphs yields the desired orthogonal representation, we have to enforce the following properties for the orthogonal representation of H. First, the orthogonal representation of H has to occupy σ and τ incidences at its poles. Second, the thick edge corresponding to a subgraph G_i is allowed to have β_i bends only if G_i has a valid (σ_i, τ_i)-orthogonal representation with β_i bends. Note that this tests the existence of an orthogonal representation without restriction to the number of bends. We will show later, how to really compute the cost function of G.

Restricting the allowed flows in the flow network such that they only represent (σ, τ)-orthogonal representations is easy. The graph H occupies σ incidences if and only if $\text{rot}(s_f) = \sigma - 3$ (where f is the outer face). As the rotation $\text{rot}(s_f)$ is represented by the flow on the corresponding vertex-face arc, we can enforce $\text{rot}(s_f) = \sigma - 3$ by setting the upper and lower bound on the corresponding arc to $\sigma - 3$. Analogously, we can ensure that H occupies τ incidences of t.

In the following we show how to restrict the number of bends of a thick edge $e_i = s_i t_i$ to the possible number of bends of the subgraph G_i it represents. Assume G_i is k_i-critical. It follows from Lemma 2.8 that G_i has gap at most k_i. Thus, the possible

values for $\text{rot}(\pi(s_i, t_i))$ can be expressed as the union of at most $k_i + 1$ intervals due to Lemma 2.9. Restricting the rotation to an interval can be easily done using capacities. However, we get $k_i + 1$ possibilities to set these capacities, and thus combining these possibilities for all thick edges results in $\prod(k_i + 1)$ flow networks.

We show that $\prod(k_i + 1)$ is in $O(2^k)$. To this end, we first show that $\sum k_i \leq k$ holds, by proving that an edge that is critical in one of the subgraphs G_i is still critical in the graph G. This is obviously true for critical edges in G_i not incident to a pole of G_i, as these inflexible edges already have endpoints with degree 4 in G_i. An edge e incident to a pole, without loss of generality s_i of G_i is critical in G_i if s_i has degree at least 2. If s_i remains a pole of G, then e is also critical with respect to G. Otherwise, s_i has degree 4 in G, which comes from the fact that the skeleton H becomes triconnected when adding the edge st.

As the 0-critical subgraphs do not play a role in the product $\prod(k_i + 1)$, we only consider the d subgraphs $G_1, \ldots, G_d$ such that G_i (for $i \in \{1, \ldots, d\}$) is k_i-critical with $k_i \geq 1$. To find the worst case, we want to maximize $\prod(k_i + 1)$ with respect to $\sum k_i \leq k$ (which is equivalent to finding a hypercuboid of dimension d with maximal volume and with fixed perimeter). We get the maximum by setting $k_i = k/d$ for all subgraphs, which results in $(k/d + 1)^d$ combinations. Substituting $k/d = x$ leads to $x^{k/x}$, which becomes maximal, when $x^{1/x}$ is maximal. Since $f(x) = x^{1/x}$ is a decreasing function, we get the worst case for $x = 1$ (when restricting x to positive integers), which corresponds to $d = k$ graphs that are 1-critical. Thus, in the worst case, we get $O(2^k)$ different combinations.

Since the flow networks have size $O(\ell)$, we can test the existence of a valid orthogonal representation of G in $O(2^k \cdot T_{\text{flow}}(\ell))$ time. However, we want to compute the cost function instead. Assume we want to test the existence of a valid orthogonal representation with a fixed number of bends β. In the following, we show how to restrict each of the flow networks to allow only flows corresponding to orthogonal representation with β bends. Then G clearly admits a valid orthogonal representation with β bends if and only if one of these flow networks admits a valid flow. The orthogonal representation of H (and thus the resulting one of G) has β bends if either $-\text{rot}(\pi(s,t)) = \beta$ or $-\text{rot}(\pi(t,s)) = \beta$. We can consider these two cases separately, resulting in a constant factor in the running time. Thus, it remains to ensure that $-\text{rot}(\pi(s,t))$ is fixed to β. This can be done by splitting the face node corresponding to the outer face such that exactly the arcs entering f from edge nodes or vertex nodes corresponding to edges and internal vertices of $\pi(s,t)$ are incident to one of the resulting nodes. Restricting the flow between the two resulting nodes representing the outer face f to β obviously enforces that $-\text{rot}(\pi(s,t)) = \beta$ holds. Thus, we could get the cost function of G by doing this for all possible values of β. However, we can get the cost function more efficiently.

Instead of fixing the value of $-\text{rot}(\pi(s,t))$ to β, we can compute maximum flows

to minimize or maximize it. Let $\text{rot}_{\min}$ and $\text{rot}_{\max}$ be the resulting minimum and maximum for $-\text{rot}(\pi(s,t))$, respectively. Note that, if $\text{rot}_{\max}$ is less than β_{low}, then there is no orthogonal representation where the number of bends are determined by the rotation along $\pi(s,t)$. Moreover, if $\text{rot}_{\min} < \beta_{\text{low}}$, we set $\text{rot}_{\min} = \beta_{\text{low}}$. It follows from basic flow theory that all values between $\text{rot}_{\min}$ and $\text{rot}_{\max}$ are also possible. Thus, after computing the two flows, we can simply set the cost function of G to 0 on that interval. To save a factor of k in the running time we do not update the cost function of G immediately, but store the interval $[\text{rot}_{\max}, \text{rot}_{\min}]$. In the end, we have $O(2^k)$ such intervals. The maximum of all upper bounds of these intervals is clearly $\beta_{\max}$ (the largest possible number of bends of G). It remains to extract the cost function of G on the interval $[\beta_{\text{low}}, \beta_{\text{low}} + k - 1]$, since the cost function of G has gap at most k (Lemma 2.8). This can be done by sorting all intervals having their lower bound in $[\beta_{\text{low}}, \beta_{\text{low}} + k - 1]$ by their lower bound. This can be done in $O(k + 2^k)$ time, since we sort $O(2^k)$ values in a range of size k. Finally, the cost function on $[\beta_{\text{low}}, \beta_{\text{low}} + k - 1]$ can be easily computed in $O(k + 2^k)$ time by scanning over this list. As this is dominated by the computation of all flows, we get an overall running time of $O(2^k \cdot T_{\text{flow}}(\ell))$. □

Lemma 2.11. *The (restricted) cost functions of a series and a parallel composition can be computed in $O(k^2 + 1)$ time if the resulting graph is k-critical.*

Proof. First, consider only the non-restricted case. Let G_1 and G_2 be the two graphs that should be composed and let G be the resulting graph. As in the rigid case, we can use flow networks to compute the cost functions of G. However, this time the flow network has constant size and thus we do not have to be so careful with the constants.

Assume G_1 and G_2 are k_1- and k_2-critical, respectively. Up to possibly a constant number, all critical edges in G_i are also critical in G, i.e., $k_i \in O(k + 1)$ (note that the "+1" is necessary for the case $k = 0$). Thus, both graphs G_1 and G_2 have a gap of size $O(k + 1)$. It follows that the possible rotations values for $\pi(s_i, t_i)$ (where s_i and t_i are the poles of G_i) are the union of $O(k + 1)$ intervals, which results in $O(k^2 + 1)$ possible combinations and thus $O(k^2 + 1)$ flow networks of constant size. Note that we get an additional constant factor by considering all possible values for the number of occupied incidences of the graphs G_i. Extracting the cost functions out of the results from the flow computation can be done analogously to the case where we had a rigid composition (proof of Lemma 2.10), which finally results in the claimed running time $O(k^2 + 1)$.

To compute the restricted cost functions, one possibly has to restrict the rotation at some vertices to -1 or 1, which can be obviously done without increasing the running time. □

Theorem 2.5. *FlexDraw for k-critical graphs can be solved in $O(2^k \cdot n \cdot T_{\text{flow}}(n))$*

Proof. By Theorem 2.3, we get an algorithm with the running time $O(n \cdot (n \cdot T_S + n \cdot T_P + T_R(n)))$, where $T_S, T_P \in O(k^2 + 1)$ (Lemma 2.11) and $T_R(\ell) = 2^k \cdot T_{\text{flow}}(\ell)$ (Lemma 2.10) holds. This obviously yields the running time $O((k^2 + 1) \cdot n^2 + 2^k \cdot n \cdot T_{\text{flow}}(n)) = O(2^k \cdot n \cdot T_{\text{flow}}(n))$. □

2.6 Conclusion

In this chapter, we have investigated the computational complexity of FlexDraw in the presence of inflexible edges. The main result is the FPT-algorithm with running time $O(2^k \cdot n \cdot T_{\text{flow}}(n))$. This is more or less the best one can hope for as the problem becomes NP-hard for $k \in O(n^\varepsilon)$ inflexible edges (for any $\varepsilon > 0$) that are evenly distributed over the graph, which includes the case that the inflexible edges form a matching.

In Chapter 3, we give an algorithm solving OptimalFlexDraw in the absence of inflexible edges (with the additional requirement that the cost functions are convex). It is a natural open question whether these results can be combined: Is there an FPT-algorithm for OptimalFlexDraw if only k edges are inflexible, i.e., if only k edges cause cost on the first bend?

One might try to apply a similar approach as in this chapter by showing that the cost functions of st-graphs are only non-convex if they contain inflexible edges. Then, when encountering a rigid composition, one could separate these non-convex cost functions into convex parts and consider all combinations of these convex parts. Unfortunately, this approach fails for reasons we discuss in the conclusion of Chapter 3.

3 Bend Minimization with Convex Bend Costs

In this chapter, we consider bend minimization in planar orthogonal drawings. More precisely, we investigate the problem OptimalFlexDraw, where each edge e has its individual cost function $\text{cost}_e\colon \mathbb{N}_0 \to \mathbb{R}$. The cost caused by the edge e with ρ bends is then $\text{cost}_e(\rho)$. This formulation is very general, in fact, it generalizes the bend-minimization problems usually considered in two directions. First, allowing arbitrary cost functions unifies multiple previous problems that usually fix the cost function, e.g., to "each bend costs 1". Second, allowing individual cost functions reflects the fact that edges have varying importance in typical applications.

This generality of the problem OptimalFlexDraw has the downside that it includes an NP-hard problem [GT01] and thus is NP-hard itself. However, we show that it can be solved efficiently if (i) the cost function of each edge is convex and (ii) each edge has positive flexibility, i.e., the first bend on each edge does not cause any cost. Our algorithm takes time $O(n \cdot T_{\text{flow}}(n))$ and $O(n^2 \cdot T_{\text{flow}}(n))$ for biconnected and connected graphs, respectively, where $T_{\text{flow}}(n)$ denotes the time to compute a minimum-cost flow in a planar network of size n with multiple sources and sinks. This result is the first polynomial-time bend-optimization algorithm for general 4-planar graphs optimizing over all embeddings. Previous work considers restricted graph classes and unit costs. Moreover, the result is optimal in the sense that omitting one of the two conditions makes OptimalFlexDraw NP-hard.

This chapter is based on joint work with Ignaz Rutter and Dorothea Wagner [BRW13].

3.1 Introduction

Tamassia [Tam87] showed that the total number of bends in orthogonal drawings can be efficiently minimized if the planar embedding of the input graph is fixed. Tamassia's algorithm is based on a reduction to a minimum-cost flow network; see the preliminaries in Section 1.4.4. In this flow network, each unit of flow on certain arcs corresponds to a bend in the resulting drawing. To solve the more general OptimalFlexDraw problem for plane graphs, one has to use the given cost functions as cost functions in the flow network. As minimum-cost flows can be efficiently computed for convex flow networks, OptimalFlexDraw can be solved efficiently if the cost functions are convex and the planar embedding of the graph is fixed; see also Figure 3.1b. It is not hard to see, that the problem becomes NP-hard if the cost functions are non-convex; see Figure 3.1a.

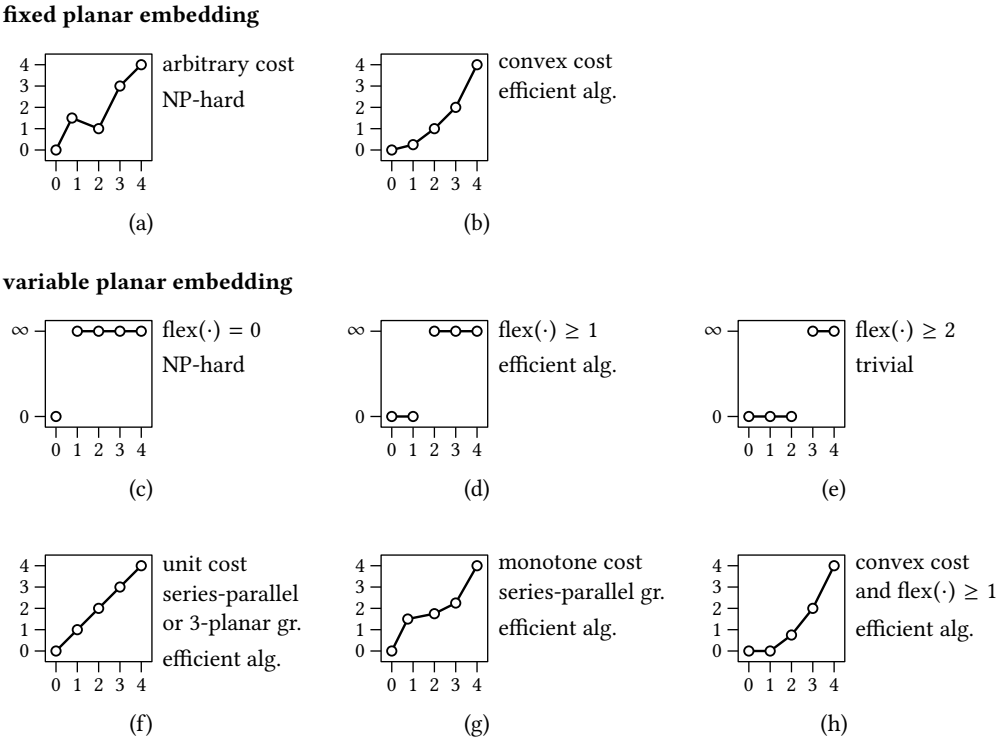


Figure 3.1: Illustration of the related work on OptimalFlexDraw. Each sub-figure represents one result (either NP-hardness or efficient algorithm) by sowing a type of cost function and maybe a restriction to a certain graph class for which the result holds. The number of bends and the resulting costs are on the x- and y-axis, respectively.

Recall from Section 2.1 that the planar embedding of the graph may have a huge impact on the number of bends; see Figure 2.1. We thus consider OptimalFlexDraw in the variable embedding setting. Although this problem is NP-hard even for the base case of testing whether there is a drawing without bends [GT01] (Figure 3.1c), it can be solved efficiently for restricted cases. The underlying decision problem FlexDraw can be efficiently solved if $\text{flex}(e) \geq 1$ for every edge e [Blä+14] and even becomes trivial if $\text{flex}(e) \geq 2$ [BK98]; see Figure 3.1d and Figure 3.1e.

Di Battista et al. [DLV98] give an algorithm minimizing the total number of bends for maximum-degree 3 and series-parallel graphs; see Figure 3.1f. In the previous chapter (see Section 2.4), we showed that one still obtains an efficient algorithm for series-parallel graphs if arbitrary monotone cost functions are allowed; see Figure 3.1g.

Contribution and Outline

The main result of this chapter is the first polynomial-time bend-minimization algorithm for general 4-planar graphs optimizing over all planar embeddings. Previous work considers only restricted graph classes and unit costs. We solve OptimalFlexDraw if (i) all cost functions are convex and (ii) the first bend is for free, i.e., the instance has positive flexibility; see Figure 3.1h. We note that not requiring positive flexibility makes the problem NP-hard as the base case of testing for cost 0 is already hard; compare the NP-hardness in Figure 3.1c and the efficient solvability in Figure 3.1d. Moreover, requiring convex cost is necessary as non-convex cost makes the problem NP-hard even for the fixed-embedding case; compare Figure 3.1a with Figure 3.1b.

Our algorithm in particular allows to efficiently minimize the total number of bends over all planar embeddings, where one bend per edge is free. Note that this is an optimization version of FlexDraw where each edges has flexibility 1, as a drawing with cost 0 exists if and only if FlexDraw has a valid solution. Moreover, as it is known that every 4-planar graph has an orthogonal representation with at most two bends per edge [BK98], our result can also be used to create such a drawing minimizing the number of edges having two bends by setting the costs for three or more bends to ∞.

To derive the algorithm for OptimalFlexDraw, we show the existence of an optimal solution with at most three bends per edge except for a single edge per block with up to four bends, confirming a conjecture of Rutter [Rut11].

Our strategy for solving OptimalFlexDraw for biconnected graphs optimizing over all planar embedding is similar to the framework presented in Chapter 2. We use dynamic programming on the SPQR-tree of the graph. Every node in the SPQR-tree corresponds to a split component (its pertinent graph) and we compute cost functions for these split components (which are st-graphs) determining the cost depending on how strongly the split component is bent. We compute such a cost function from the cost functions of the children using a flow network similar to the one described by Tamassia [Tam87].

As computing flows with minimum cost is NP-hard for non-convex costs we need to ensure that not only the cost functions of the edges but also the cost functions of the split components we compute are convex. However, this is not true at all, see Figure 3.2 for an example. This is not even true if every edge can have a single bend for free and then has to pay cost 1 for every additional bend, see Figure 3.2c. To solve this problem, we essentially show that it is sufficient to compute the cost functions on the small interval $[0,3]$. We can then show that the cost functions we compute are (almost) always convex on this interval.

We first consider the decision problem FlexDraw for the case that the planar embedding is fixed in Section 3.2. In this restricted setting we are able to prove the existence of valid drawings with special properties. Bläsius et al. [Blä+14] show that

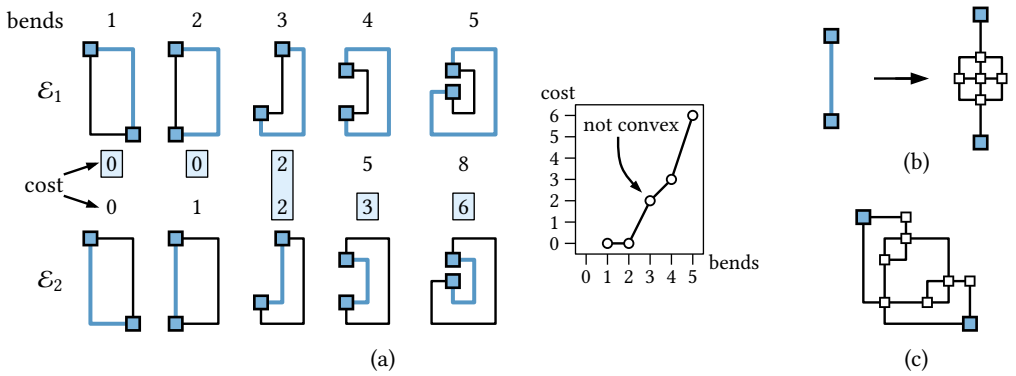


Figure 3.2: (a) Two parallel edges, the black edge has one bend for free, every additional bend costs 1, the blue edge has two bends for free, every additional bend costs 2. Whether embedding $\mathcal{E}_1$ or $\mathcal{E}_2$ is better depends on the number of bends. The minimum (marked by blue boxes) yields a non-convex cost function. (b) The non-convexity in (a) does not rely on multiple edges; the blue edge could be replaced by the shown gadget where each edge of the gadget has one bend for free and every additional bend costs 2. (c) This graph has a non-convex cost function even if every edge has one bend for free and each additional bend costs 1.

"rigid" graphs do not exist in this setting in the sense that a drawing that is bent strongly can be unwound under the assumption that the flexibility of every edge is at least 1. In other words this shows that *st*-graphs with positive flexibility behave similar to single edges with positive flexibility. We present a more elegant proof yielding a stronger result that can then be used to reduce the number of bends of every edge down to three (at least for biconnected graphs and except for a single edge on the outer face).

In Section 3.3 we extend this to split components of a graph. More precisely, we show that in a biconnected graph the split components corresponding to the nodes in its SPQR-tree can be assumed to have only up to three bends. In Section 3.4 we show that these results for the decision problem FlexDraw can be extended to the optimization problem OptimalFlexDraw. With this result we are able to drop the fixed planar embedding (Section 3.5). We first consider biconnected graphs in Section 3.5.1 and compute cost functions on the interval $[0,3]$, which can be shown to be (almost always) convex on that interval, bottom up in the SPQR-tree. In Section 3.5.2 we extend this result to connected graphs using the BC-tree (see the preliminaries in Section 1.4.3 for a definition).

3.2 Valid Drawings with Fixed Planar Embedding

In this section we consider the problem FlexDraw for the case that the planar embedding is fixed. We show that the existence of a valid orthogonal representation implies

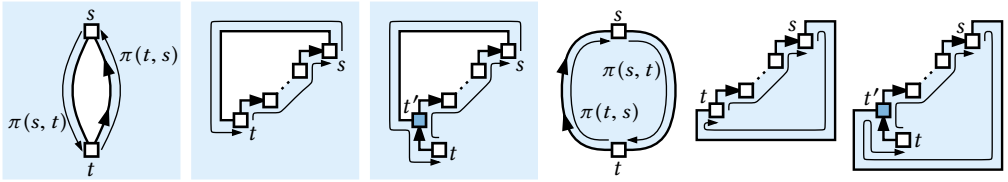


Figure 3.3: A strictly directed path from t to s has a lower bound for its rotation. This yields upper bounds for paths from s to t (Lemma 3.1).

the existence of a valid orthogonal representation with special properties. We first show the following. Given a biconnected 4-planar graph with positive flexibility and an orthogonal representation $\mathcal{R}$ such that two vertices s and t lie on the outer face f, then the rotation along $\pi_f(s,t)$ can be reduced by 1 if it is at least 0. This result is a key observation for the algorithm solving the decision problem FlexDraw [Blä+14]. It in a sense shows that "rigid" graphs that have to bent strongly do not exists. This kind of graphs play an important role in the NP-hardness proof of 0-embeddability by Garg and Tamassia [GT01]. Moreover, we show the existence of a valid orthogonal representation $\mathcal{R}'$ inducing the same planar embedding and having the same angles around vertices as $\mathcal{R}$ such that every edge has at most three bends in $\mathcal{R}'$, except for a single edge on the outer face with up to five bends. If we allow to change the embedding slightly, this special edge has only up to four bends.

Let G be a 4-planar graph with positive flexibility and valid orthogonal representation $\mathcal{R}$, and let e be an edge. If the number of bends of e equals its flexibility, we orient e such that its bends are right bends. Otherwise, e remains undirected. As G may include directed and undirected edges, we say that G is a *mixed* graph. A path $\pi = (v_0, \dots, v_k)$ in a mixed graph G is a *directed* path if the edge connecting v_{i-1} with v_i is either undirected or directed from v_{i-1} to v_i for $i = 1, \dots, k$. A path containing only undirected edges can be seen as directed path for both possible directions. The path π is *strictly directed* if it is directed and does not contain undirected edges. These definitions directly extend to *(strictly) directed* cycles.

Given a (strictly) directed cycle C the terms left(C) and right(C) denote the set of edges and vertices of G lying to the left and right of C, respectively, with respect to the orientation of C. A cut $(U, V \setminus U)$ is said to be *directed* from U to $V \setminus U$, if every edge $\{u, v\}$ with $u \in U$ and $v \in V \setminus U$ is either directed from u to v or undirected. According to the above definitions a cut is *strictly directed* from U to $V \setminus U$ if it is directed and contains no undirected edges. Before we show how to unwind an orthogonal representation that is bent strongly we need the following technical lemma.

Lemma 3.1. *Let G be a graph with positive flexibility and vertices s and t such that $G + st$ is biconnected and 4-planar. Let further $\mathcal{R}$ be a valid orthogonal representation*

with s and t incident to the common face f such that $\pi_f(t,s)$ is strictly directed from t to s. Then the following holds.

(1) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \leq -3$ *if f is the outer face and G does not consist of a single path*

(2) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \leq -1$ *if f is the outer face*

(3) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \leq 5$

Proof. We first consider the case where f is the outer face (Figure 3.3a), i.e., cases (1) and (2). Due to the fact that $\pi_f(t,s)$ is strictly directed from t to s and the flexibility of every edge is positive, each edge on $\pi_f(t,s)$ has rotation at least 1. Moreover, the rotations at vertices along the path $\pi_f(t,s)$ are at least -1 since $\pi_f(t,s)$ is simple as $G+st$ is biconnected. Since the number of internal vertices on a path is one less than the number of edges this yields $\text{rot}(\pi_f(t,s)) \geq 1$; see Figure 3.3b. If G consists of a single path this directly yields $\text{rot}(\pi_f(s,t)) \leq -1$ and thus concludes case (2). For case (1) first assume that the degrees of s and t are not 1 (Figure 3.3b), i.e., $\text{rot}(s_f), \text{rot}(t_f) \in \{-1,0,1\}$ holds. Since f is the outer face the equation $\text{rot}(\pi_f(s,t)) + \text{rot}(t_f) + \text{rot}(\pi_f(t,s)) + \text{rot}(s_f) = -4$ holds and directly implies the desired inequality $\text{rot}(\pi_f(s,t)) \leq -3$. In the case that for example t has degree 1 (and $\deg(s) > 0$), we have $\text{rot}(t_f) = -2$ and $\text{rot}(s_f) \in \{-1,0,1\}$, thus the considerations above only yield $\text{rot}(\pi_f(s,t)) \leq -2$. However, in this case there necessarily exists a vertex t' where the paths $\pi_f(s,t)$ and $\pi_f(t,s)$ split, as illustrated in Figure 3.3c. More precisely, let t' be the first vertex on $\pi_f(s,t)$ that also belongs to $\pi_f(t,s)$. Obviously, the degree of t' is at least 3 and thus $\text{rot}(t'_f)$ (with respect to the path $\pi_f(t,s)$) is at least 0. Hence we obtain the stronger inequality $\text{rot}(\pi_f(t,s)) \geq 2$ yielding the desired inequality $\text{rot}(\pi_f(s,t)) \leq -3$. If s and t both have degree 1 we cannot only find the vertex t' but also the vertex s' where the paths $\pi_f(s,t)$ and $\pi_f(t,s)$ split. Since $G + st$ is biconnected these two vertices are distinct and the estimation above works, finally yielding $\text{rot}(\pi_f(s,t)) \leq -3$.

If f is an internal face (Figure 3.3d), i.e., case (3) applies, we start with the equation $\text{rot}(\pi_f(s,t)) + \text{rot}(t_f) + \text{rot}(\pi_f(t,s)) + \text{rot}(s_f) = 4$. First we consider the case that neither t nor s have degree 1. Thus, $\text{rot}(t_f), \text{rot}(s_f) \in \{-1,0,1\}$. With the same argument as above we obtain $\text{rot}(\pi_f(t,s)) \geq 1$ and hence $\text{rot}(\pi_f(s,t)) \leq 5$; see Figure 3.3e. Now assume that t has degree 1 and s has larger degree. Then $\text{rot}(t_f) = -2$ holds and the above estimation does not work anymore. Again, at some vertex t' the paths $\pi_f(t,s)$ and $\pi_f(s,t)$ split as illustrated in Figure 3.3f. Obviously, the degree of t' needs to be greater than 2 and thus $\text{rot}(t'_f)$ is at least 0. This yields $\text{rot}(\pi_f(t,s)) \geq 2$ in the case that $\deg(t) = 1$, compensating $\text{rot}(t_f) = -2$ (instead of $\text{rot}(t_f) \geq -1$ in the other case). To sum up, we obtain the desired inequality $\text{rot}(\pi_f(s,t)) \leq 5$. The case $\deg(s) = \deg(t) = 1$ works analogously. □

The *flex graph* $G^{\times}_{\mathcal{R}}$ of G with respect to a valid orthogonal representation $\mathcal{R}$ is defined to be the dual graph of G such that the dual edge $e^{\star}$ is undirected if e is undirected,

otherwise it is directed from the face right of e to the face left of e. Figure 3.4a shows an example graph with an orthogonal drawing together with the corresponding flex graph. Assume we have a simple directed cycle C in the flex graph. Then *bending* along this cycle yields a new valid orthogonal representation $\mathcal{R}'$ which is defined as follows. Let $e^\star = (f_1, f_2)$ be an edge contained in C dual to e. Then we decrease $\text{rot}(e_{f_1})$ and increase $\text{rot}(e_{f_2})$ by 1. It can be easily seen that the necessary properties for $\mathcal{R}'$ to be an orthogonal representation are satisfied. Obviously, $\text{rot}_{\mathcal{R}'}(e_{f_1}) = -\text{rot}_{\mathcal{R}'}(e_{f_2})$ holds and rotations at vertices did not change. Moreover, the rotation around a face f does not change since f is either not contained in C or it is contained in C, but then it has exactly one incoming and exactly one outgoing edge. Note that bending along a cycle in the flex graph preserves the planar embedding of G and for every vertex the rotations in all incident faces. The following lemma shows that a high rotation along a path $\pi_f(s,t)$ for two vertices s and t sharing the face f can be reduced by 1 using a directed cycle in the flex graph.

Lemma 3.2. *Let G be a biconnected 4-planar graph with positive flexibility, a valid orthogonal representation $\mathcal{R}$ and s and t on a common face f. The flex graph $G^\times_{\mathcal{R}}$ contains a directed cycle C such that $f \in C$, $s \in \text{left}(C)$ and $t \in \text{right}(C)$, if one of the following conditions holds.*

(1) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq -2$, *$f$ is the outer face and $\pi_f(s,t)$ is not strictly directed from t to s*

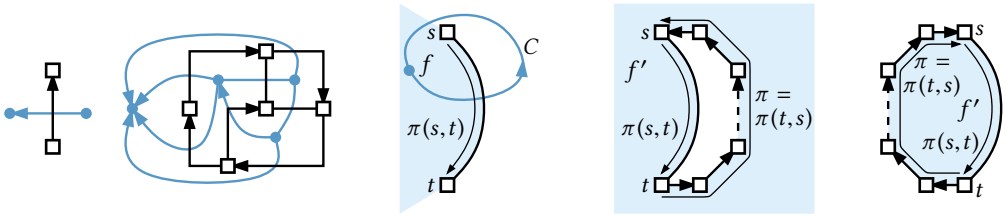
(2) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq 0$ *and f is the outer face*

(3) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq 6$

Proof. Figure 3.4b shows the path $\pi_f(s,t)$ together with the desired cycle C. Due to the duality of a cycle in the dual and a cut in the primal graph a directed cycle C in $G^\times_{\mathcal{R}}$ having s and t to the left and to the right of C, respectively, induces a directed cut in G that is directed from s to t and vice versa. Recall that directed cycles and cuts may also contain undirected edges. Assume for contradiction that such a cycle C does not exist.

Claim 1. *The graph G contains a strictly directed path π from t to s.*

Every cut (S,T) with $T = V \setminus S$, $s \in S$ and $t \in T$ separating s from t must contain an edge that is directed from T to S, otherwise this cut would correspond to a cycle C in the flex graph that does not exist by assumption. Let T be the set of vertices in G that can be reached by strictly directed paths from t. If T contains s we found the path π strictly directed from t to s. Otherwise, (S,T) with $S = V \setminus T$ is a cut separating S from T and there cannot be an edge that is directed from a vertex in T to a vertex in S which is a contradiction, and thus the path π strictly directed from t to s exists, which concludes the proof of the claim.



(a) (b) (c) (d)

Figure 3.4: (a) An orthogonal representation and the corresponding flex graph (blue) where every edge has flexibility 1. (b, c, d) Illustration of Lemma 3.2.

Let G' be the subgraph of G induced by the paths π and $\pi_f(s,t)$ together with the orthogonal representation $\mathcal{R}'$ induced by $\mathcal{R}$.

We first consider case (1). Let f' be the outer face of the orthogonal representation $\mathcal{R}'$. Obviously, $\pi_{f'}(s,t) = \pi_f(s,t)$ and $\pi = \pi_{f'}(t,s)$ holds, see Figure 3.4c. Moreover, the graph $G' + st$ is biconnected and G' does not consist of a single path since $\pi_{f'}(s,t)$ and $\pi_{f'}(t,s)$ are different due to the assumption that $\pi_f(s,t)$ is not strictly directed from t to s. Since $\pi_{f'}(t,s)$ is strictly directed from t to s we can use Lemma 3.1(1) yielding $\text{rot}_{\mathcal{R}'}(\pi_{f'}(s,t)) \leq -3$ and thus $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \leq -3$, which is a contradiction.

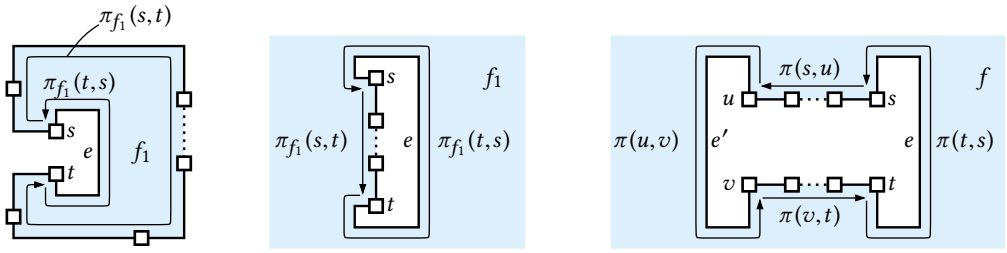
For case (2) exactly the same argument holds except for the case where the strictly directed path π is the path $\pi_f(s,t)$ strictly directed from t to s. In this case we have to use Lemma 3.1(2) instead of Lemma 3.1(1) yielding $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \leq -1$, which is again a contradiction.

In case (3) the subgraph G' of G induced by the two paths π and $\pi_f(s,t)$ again contains s and t on a common face f', which may be the outer or an inner face, see Figure 3.4c and Figure 3.4d, respectively. In both cases we obtain $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \leq 5$ due to Lemma 3.1(3), which is a contradiction. □

Lemma 3.2 directly yields the following corollary, showing that graphs with positive flexibility behave very similar to single edges with positive flexibility.

Corollary 3.1. *Let G be a graph with positive flexibility and vertices s and t such that $G + st$ is biconnected and 4-planar. Let further $\mathcal{R}$ be a valid orthogonal representation with s and t on the outer face f such that $\rho = \text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq 0$. For every rotation $\rho' \in [-1, \rho]$ there exists a valid orthogonal representation $\mathcal{R}'$ with $\text{rot}_{\mathcal{R}'}(\pi_f(s,t)) = \rho'$.*

Proof. For the case that G itself is biconnected, the claim follows directly from Lemma 3.2(2), since we can reduce the rotation along $\pi_f(s,t)$ stepwise by 1, starting with the orthogonal representation $\mathcal{R}$, until we reach a rotation of -1. For the case that G itself is not biconnected we add the edge $\{s,t\}$ to the orthogonal representation $\mathcal{R}$ such that the path $\pi_f(s,t)$ does not change, i.e., $\pi_f(t,s)$ consists of the new edge $\{s,t\}$. Again Lemma 3.2(2) can be used to reduce the rotation stepwise down to -1. □



(a) (b) (c)

Figure 3.5: Reducing the number of bends on edges (Theorem 3.1).

As edges with many bends imply the existence of paths with high rotation, we can use Lemma 3.2 to successively reduce the number of bends of every edge down to three, except for a single edge on the outer face. Since we only bend along cycles in the flex graph, neither the embedding nor the angles around vertices are changed.

Theorem 3.1. *Let G be a biconnected 4-planar graph with positive flexibility, having a valid orthogonal representation. Then G has a valid orthogonal representation with the same planar embedding, the same angles around vertices and at most three bends per edge, except for at most one edge on the outer face with up to five bends.*

Proof. In the following we essentially pick an edge with more than three bends, reduce the number of bends by one and continue with the next edge. After each of these reduction steps we set the flexibility of every edge down to $\max\{\rho, 1\}$, where ρ is the number of bends it currently has. This ensures that in the next step the number of bends of each edge either is decreased, remains as it is or is increased from zero to one.

We start with an edge $e = \{s,t\}$ that is incident to two faces f_1 and f_2 and has more than three bends. Due to the fact that we traverse inner faces in clockwise and the outer face in counter-clockwise direction, the edge e forms in one of the two faces the path from s to t and in the other face the path from t to s. Assume without loss of generality that $\pi_{f_1}(t,s)$ and $\pi_{f_2}(s,t)$ are the paths on the boundary of f_1 and f_2, respectively, that consist of e. Note that $\operatorname{rot}(\pi_{f_1}(t,s)) = -\operatorname{rot}(\pi_{f_2}(s,t))$ holds and we assume that $\operatorname{rot}(\pi_{f_1}(t,s))$ is not positive. As e was assumed to have more than three bends, the inequality $\operatorname{rot}(\pi_{f_1}(t,s)) \leq -4$ holds. We distinguish between the two cases that f_1 is an inner or the outer face. We first consider the case that f_1 is an inner face; Figure 3.5a illustrates this situation for the case where e has four bends. Then the rotations around the face f_1 sum up to 4. As the rotations at the vertices s and t can be at most 1, we obtain $\operatorname{rot}(\pi_{f_1}(s,t)) \geq 6$. Thus we can apply Lemma 3.2(3) to reduce the rotation of $\pi_{f_1}(s,t)$ by bending along a cycle in the flex graph that contains f_1 and separates s from t. Obviously, this increases the rotation along $\pi_{f_1}(t,s)$ by 1 and thus reduces the number of bends of e by 1.

For the case that f_1 is the outer face we first ignore the case where e has four or five bends and show how to reduce the number of bends to five; Figure 3.5b shows the case where e has six bends. Thus the inequality $\text{rot}(\pi_{f_1}(t,s)) \leq -6$ holds. As the rotations around the outer face f_1 sum up to -4 and the rotations at the vertices s and t are at most 1, the rotation along $\pi_{f_1}(s,t)$ must be at least 0. Thus we can apply Lemma 3.2(2) to reduce the rotation of $\pi_{f_1}(s,t)$ by 1, increasing the rotation along $\pi_{f_1}(t,s)$, and thus reducing the number of bends of e by one.

Finally, we obtain an orthogonal representation having at most three bends per edge except for some edges on the outer face with four or five bends having their negative rotation in the outer face. If there is only one of these edges left we are done. Otherwise let $e = \{s,t\}$ be one of the edges with $\text{rot}(\pi_f(t,s)) \in \{-5,-4\}$, where f is the outer face. Then the inequality $\text{rot}(\pi_f(s,t)) \geq -2$ holds by the same argument as before and we can apply Lemma 3.2(1) to reduce the rotation, if we can ensure that $\pi_f(s,t)$ is not strictly directed from t to s. To show that, we make use of the fact that $\pi_f(s,t)$ contains an edge $e' = \{u,v\}$ with at least four bends due to the assumption that e was not the only edge with more than three bends. Assume without loss of generality that u occurs before v on $\pi_f(s,t)$, thus $\pi_f(s,t)$ splits into the three parts $\pi_f(s,u)$, $\pi_f(u,v)$ and $\pi_f(v,t)$. Recall that $\text{rot}(\pi_f(s,t)) \geq -2$ holds and thus $\text{rot}(\pi_f(s,u)) + \text{rot}(u) + \text{rot}(\pi_f(u,v)) + \text{rot}(v) + \text{rot}(\pi_f(v,t)) \geq -2$. As the rotation at the vertices u and v is at most 1 and the rotation of $\pi_f(u,v)$ at most -4 it follows that $\text{rot}(\pi_f(s,u)) + \text{rot}(\pi_f(v,t)) \geq 0$. Figure 3.5c illustrates the situation for the case where e and e' have four bends and $\text{rot}(\pi_f(s,u)) = \text{rot}(\pi_f(v,t)) = 0$. Note that at least one of the two paths is not degenerate in the sense that $s \neq u$ or $v \neq t$, otherwise the total rotation around the outer face would be at most -6, which is a contradiction. Assume without loss of generality that $\text{rot}(\pi_f(s,u)) \geq 0$. It follows that $\pi_f(s,u)$ cannot be strictly directed from u to s and since $\pi_f(s,u)$ is a subpath of $\pi_f(s,t)$ the path $\pi_f(s,t)$ cannot be strictly directed from t to s. This finally shows that we can use part (1) of Lemma 3.2 implying that we can find a valid orthogonal representation such that at most a single edge with four or five bends remains, whereas all other edges have at most three bends. □

If we allow the embedding to be changed slightly, we obtain an even stronger result. Assume the edge e lying on the outer face has more than three bends. If e has five bends, we can reroute it in the opposite direction around the rest of the graph, i.e., we can choose the internal face incident to e to be the new outer face. In the resulting drawing e has obviously only three bends. Thus the following result directly follows from Theorem 3.1.

Corollary 3.2. *Let G be a biconnected 4-planar graph with positive flexibility having a valid orthogonal representation. Then G has a valid orthogonal representation with at most three bends per edge except for possibly a single edge on the outer face with four bends.*

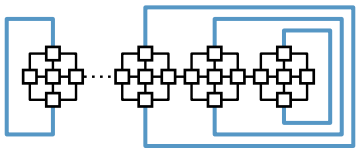


Figure 3.6: An instance of FlexDraw requiring linearly many edges to have four bends. Flexibilites are 1 except for the blue edges with flexibility 4.

Note that Corollary 3.2 is restricted to biconnected graphs. For general graphs it implies that each block contains at most a single edge with up to four bends. Figure 3.6 illustrates an instance of FlexDraw with linearly many blocks and linearly many edges that are required to have four bends, showing that Corollary 3.2 is tight.

Theorem 3.1 implies that it is sufficient to consider the flexibility of every edge to be at most 5, or in terms of costs we want to optimize, it is sufficient to store the cost function of an edge only in the interval $[0,5]$. However, there are two reasons why we need a stronger result. First, we want to compute cost functions of split components and thus we have to limit the number of "bends" they can have (see the next section for a precise definition of bends for split components). Second, as mentioned in the introduction (see Figure 3.2) the cost function of a split component may already be non-convex on the interval $[0,5]$. Fortunately, the second reason is not really a problem since there may be at most a single edge with up to five bends, all remaining edges have at most three bends and thus we only need to consider their cost functions on the interval $[0,3]$.

In the following section we focus on dealing with the first problem and strengthen the results so far presented by extending the limitation on the number of bends to split components. Note that a split pair inside an inner face of G with a split component H having a rotation less than -3 on its outer face implies a rotation of at least 6 in some inner face of G. Thus, we can again apply Lemma 3.2(3) to reduce the rotation showing that split components and single edges can be handled similarly. However, by reducing the rotation for one split component, we cannot avoid that the rotation of some other split component is increased. For single edges we did that by reducing the flexibility to the current number of bends. In the following section we extend this technique by defining a flexibility not only for edges but also for split components. We essentially show that all results we presented so far still apply, if we allow this kind of extended flexibilities.

3.3 Flexibility of Split Components and Nice Drawings

Let G be a biconnected 4-planar graph with SPQR-tree $\mathcal{T}$ and let $\mathcal{T}$ be rooted at some node τ. Recall that we do not require τ to be a Q-node. Let μ be a node of $\mathcal{T}$ that is not the root τ. Then μ has a unique parent and $\text{skel}(\mu)$ contains a unique virtual edge $\varepsilon = \{s,t\}$ that is associated with this parent. We call the split-pair $\{s,t\}$ a *principal split pair* and the pertinent graph $\text{pert}(\mu)$ with respect to the chosen root a *principal*

split component. The vertices s and t are the *poles* of this split component. Note that a single edge is also a principal split component except for the case that its Q-node is chosen to be the root. A planar embedding of G is represented by $\mathcal{T}$ with the root τ if the embedding of each skeleton has the edge associated with the parent on the outer face.

Let $\mathcal{R}$ be a valid orthogonal representation of G such that the planar embedding of $\mathcal{R}$ is represented by $\mathcal{T}$ rooted at τ. Consider a principal split component H with respect to the split pair $\{s,t\}$ and let $\mathcal{S}$ be the orthogonal representation of H induced by $\mathcal{R}$. Note that the poles s and t are on the outer face f of $\mathcal{S}$. We define $\max\{|\operatorname{rot}_{\mathcal{S}}(\pi_f(s,t))|, |\operatorname{rot}_{\mathcal{S}}(\pi_f(t,s))|\}$ to be the *number of bends* of the split component H. Note that this is a straightforward extension of the term *bends* as it is used for edges. With this terminology we can assign a *flexibility* $\operatorname{flex}(H)$ to a principal split component H and we define the orthogonal representation $\mathcal{R}$ of G to be *valid* if and only if H has at most $\operatorname{flex}(H)$ bends. We say that the graph G has *positive flexibility* if the flexibility of every principal split component is at least 1, which is straightforward extension of the original notion.

We define a valid orthogonal representation of G to be *nice* if it is tight and if there is a root τ of the SPQR-tree such that every principal split component has at most three bends and the edge corresponding to τ in the case that τ is a Q-node has at most five bends. The main result of this section will be the following theorem, which directly extends Theorem 3.1.

Theorem 3.2. *Every biconnected 4-planar graph with positive flexibility having a valid orthogonal representation has an orthogonal representation with the same planar embedding and the same angles around vertices that is nice with respect to at least one node chosen as root of its SPQR-tree.*

Before we prove Theorem 3.2 we need to make some additional considerations. In particular we need to extend the flex-graph such that it takes the flexibilities of principal split components into account. The extended version of the flex graph can then be used to obtain a result similar to Lemma 3.2, which was the main tool to prove Theorem 3.1. Another difficulty is that it depends on the chosen root which split components are principal split components. For the moment we avoid this problem by choosing an arbitrary Q-node to be the root of the SPQR-tree $\mathcal{T}$. Thus we only have to care about the flexibilities of the principal split components with respect to the chosen root. One might hope that the considerations we make for the flex-graph in the case of a fixed root still work, if we consider the principal split components with respect to all possible roots at the same time. However, this fails as we will see later, making it necessary to consider internal vertices as the root.

Assume that the SPQR-tree $\mathcal{T}$ of G is rooted at the Q-node corresponding to an arbitrary chosen edge. Let H be a principal split component with respect to the chosen root with the poles s and t. In the embedding of G the outer face f of H splits into two

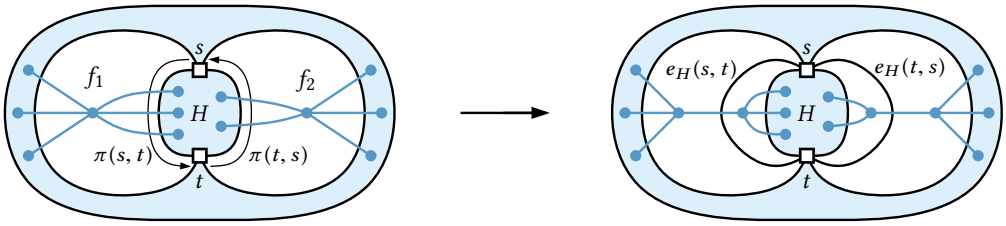


Figure 3.7: Augmentation of G with the safety edges $e_H(s,t)$ and $e_H(t,s)$.

faces f_1 and f_2, where the path $\pi_f(s,t)$ is assumed to lie in f_1 and $\pi_f(t,s)$ is assumed to lie in f_2, i.e., $\pi_{f_1}(s,t) = \pi_f(s,t)$ and $\pi_{f_2}(t,s) = \pi_f(t,s)$. We augment G by inserting the edge $\{s,t\}$ twice, embedding one of them in f_1 and the other in f_2. We denote the edge $\{s,t\}$ inserted into the face f_1 by $e_H(s,t)$ and the edge inserted into f_2 by $e_H(t,s)$. Figure 3.7 illustrates this process and shows how the dual graph of G changes. We call the new edges $e_H(s,t)$ and $e_H(t,s)$ *safety edges* and define the *extended flex graph* $G^\times$ as before, ignoring that some edges have a special meaning. To simplify notation we often use the term flex graph, although we refer to the extended flex graph. Note that every cycle in the flex graph that separates s from t and thus crosses $\pi(s,t)$ and $\pi(t,s)$ needs to also cross the safety edges $e_H(s,t)$ and $e_H(t,s)$. Thus we can use the safety edges to ensure that the flex graph respects the flexibility of H by orienting them if necessary. More precisely, we orient the safety edge $e_H(s,t)$ from t to s if $\text{rot}(\pi(s,t)) = -\text{flex}(H)$ and similarly $e_H(t,s)$ from s to t if $\text{rot}(\pi(t,s)) = -\text{flex}(H)$. This ensures that the rotations along $\pi(s,t)$ and $\pi(t,s)$ cannot be reduced below $-\text{flex}(H)$ by bending along a cycle in the flex graph. Moreover, $\text{rot}(\pi(s,t))$ cannot be increased above $\text{flex}(H)$ as otherwise $\text{rot}(\pi(t,s))$ has to be below $-\text{flex}(H)$ and vice versa. To sum up, we insert the safety edges next to the principal split component H and orient them if necessary to ensure that bending along a cycle in the flex graph respects not only the flexibilities of single edges but also the flexibility of the principal split component H.

Since adding the safety edges for the graph H is just a technique to respect the flexibility of H by bending along a cycle in the flex graph, we do not draw them. Note that the augmented graph does not have maximum degree 4 anymore but this is not a problem since we do not draw the safety edges. However, we formally assign an orthogonal representation to the safety edges by essentially giving them the shape of the paths they "supervise". More precisely, the edges $e_H(s,t)$ and $e_H(t,s)$ have the same rotations as the paths $\pi(s,t)$ and $\pi(t,s)$ on the outer face of H, respectively. Moreover, the angles at the vertices s and t are also assumed to be the same as for these two paths.

As we do not only want to respect the flexibility of a single split component, we add the safety edges for each of the principal split components at the same time. Note that the augmented graph remains planar as we only add the safety edges for the

principal split components with respect to a single root. It follows directly that the considerations above still work, which would fail if the augmented graph was non-planar. This is the reason why we cannot consider the principal split components with respect to all roots at the same time. The following lemma directly extends Lemma 3.2 to the case where the extended flex graph is considered.

Lemma 3.3. *Let G be a biconnected 4-planar graph with positive flexibility, a valid orthogonal representation $\mathcal{R}$ and s and t on a common face f. The extended flex graph $G^{\times}_{\mathcal{R}}$ contains a directed cycle C such that $f \in C$, $s \in \text{left}(C)$ and $t \in \text{right}(C)$, if one of the following conditions holds.*

(1) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq -2$, *$f$ is the outer face and $\pi_f(s,t)$ is not strictly directed from t to s*

(2) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq 0$ *and f is the outer face*

(3) $\text{rot}_{\mathcal{R}}(\pi_f(s,t)) \geq 6$

Proof. As in the proof of Lemma 3.2 we assume for contradiction that the cycle C does not exists, yielding a strictly directed path from t to s in G. This directly yields the claim, if we can apply Lemma 3.1 as before. The only difference to the situation before is that the directed path from t to s may contain some of the safety edges. However, by definition a safety edge $e_H(u,v)$ is directed from v to u if and only if $\text{rot}(\pi(u,v)) = -\text{flex}(H)$. As $\text{flex}(H)$ is positive $\text{rot}(\pi(u,v))$ has to be negative and thus the rotation along $e_H(u,v)$ when traversing it from v to u is at least 1. Thus, it does not make a difference whether the directed path from t to s consists of normal edges or may contain safety edges. Hence, Lemma 3.1 extends to the augmented graph containing the safety edges, which concludes the proof. □

Now we are ready to prove Theorem 3.2. To improve readability we state it again.

Theorem 3.2. *Every biconnected 4-planar graph with positive flexibility having a valid orthogonal representation has an orthogonal representation with the same planar embedding and the same angles around vertices that is nice with respect to at least one node chosen as root of its SPQR-tree.*

Proof. Let $\mathcal{R}$ be a valid orthogonal representation of G. We assume without loss of generality that $\mathcal{R}$ is tight. Since the operations we apply to $\mathcal{R}$ in the following do not affect the angles around vertices, the resulting orthogonal representation is also tight. Thus it remains to enforce the more interesting condition for orthogonal representations to be nice, i.e., reduce the number of bends of principal split components down to three. As mentioned before, the SPQR-tree $\mathcal{T}$ of G is initially rooted at an arbitrary Q-node. Let e_{ref} be the corresponding edge. As in the proof of Theorem 3.1 we start with an arbitrary principal split component H with more than three bends. Then one of the two paths in the outer face of H has rotation less than -3 and we have the same

situation as for a single edge, i.e., we can apply Lemma 3.3 to reduce the rotation of the opposite site and thus reduce the number of bends of H by one. Afterwards, we can set the flexibility of H down to the new number of bends ensuring that it is not increased later on. However, this only works if the negative rotation of the split component H lies in an inner face of G. On the outer face we can only increase to a rotation of -5 yielding an orthogonal representation such that every principal split component has at most three bends, or maybe four or five bends, if it has its negative rotation in the outer face. Note that this is essentially the same situation we also had in the proof of Theorem 3.1. In the following we show similarly that the number of bends can be reduced further, until either a unique innermost principal split component (where innermost means minimal with respect to inclusion) or the reference edge e_{ref} may have more than three bends.

First assume that e_{ref} has more than three, i.e., four or five, bends and that there is a principal split component H with more than three bends having its negative rotation on the outer face. Let $\{s,t\}$ be the corresponding split pair and let without loss of generality $\pi_f(t,s)$ be the path along H with rotation less than -3 where f is the outer face. Then the path $\pi_f(s,t)$ contains the edge $e_{\text{ref}} = \{u,v\}$, otherwise H would not be a principal split component. Moreover, $\text{rot}(\pi_f(t,s)) \leq -4$ implies that $\text{rot}(\pi_f(s,t)) \geq -2$ holds. As in the proof of Theorem 3.1 (compare with Figure 3.5c) the path $\pi_f(s,t)$ splits into the paths $\pi_f(s,u)$, $\pi_f(u,v)$ and $\pi_f(v,t)$. Since $\pi_f(u,v)$ consists of the single edge e_{ref} with more than three bends $\text{rot}(\pi_f(u,v)) \leq -4$ holds, implying that the rotation along $\pi_f(s,u)$ or $\pi_f(v,t)$ is greater or equal to 0. This shows that $\pi_f(s,t)$ cannot be strictly directed from t to s and thus we can apply Lemma 3.3(1) to reduce the number of bends H has. Finally, there is no principal split component with more than three bends left and the reference edge e_{ref} has at most five bends, which concludes this case.

In the second case, e_{ref} has at most three bends. We show that if there is more than one principal split component with more than three bends, then they hierarchically contain each other. Assume that the number of bends of no principal split component that has more than three bends can be reduced further. Assume further there are two principal split components H_1 and H_2 with respect to the split pairs $\{s_1,t_1\}$ and $\{s_2,t_2\}$ that do not contain each other, i.e, without loss of generality the vertices t_1, s_1, t_2 and s_2 occur in this order around the outer face f when traversing it in counter-clockwise direction and $\pi_f(t_1,s_1)$ and $\pi_f(t_2,s_2)$ belong to H_1 and H_2 respectively. Analogous to the case where e_{ref} has more than three bends we can show that Lemma 3.3(1) can be applied to reduce the number of bends of H_1, which is a contradiction. Thus, either H_1 is contained in H_2 or the other way round. This shows that there is a unique principal split component H that is minimal with respect to inclusion having more than three bends. Due to the inclusion property, all nodes in the SPQR-tree corresponding to the principal split components with more than three bends lie on the path between the

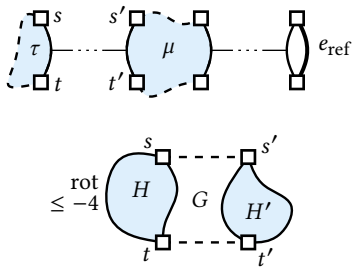


Figure 3.8: The path between the new and the old root in the SPQR-tree containing μ (top). The whole graph G containing the principal split component H' corresponding to μ with respect to the new root and the principal split component H of the new root with respect to the old root (bottom).

current root and the node corresponding to H. We denote the node corresponding to H by τ and choose τ to be the new root of the SPQR-tree $\mathcal{T}$. Since the principal split components depend on the root chosen for $\mathcal{T}$ some split components may no longer be principal and some may become principal due to rerooting. Our claim is that all principal split components with more than three bends are no longer principal after rerooting and furthermore that all split components becoming principal can be enforced to have at most three bends.

First note that the principal split component corresponding to a node μ in the SPQR-tree changes if and only if μ lies on the path between the old and the new root, i.e., between τ and the Q-node corresponding to e_{ref}. Since all principal split components (with respect to the old root) that have more than three bends also lie on this path, all these split components are no longer principal (with respect to the new root). It remains to deal with the new principal split components corresponding to the nodes on this path. Note that the new root τ itself has no principal split component associated with it. Let $\mu \neq \tau$ be a node on the path between the new and the old root and let H' be the new principal split component corresponding to μ with the poles s' and t'. Recall that H is the former principal split component corresponding to the new root τ with the poles s and t. Note that H of course is still a split component, although it is not principal anymore. Figure 3.8 illustrates this situation. Now assume that H' has more than three bends. Then there are two possibilities, either it has its negative rotation on the outer face or in some inner face. If only the latter case arises we can easily reduce the number of bends down to three as we did before. In the remaining part of the proof we show that the former case cannot arise due to the assumption that the number of bends of H cannot be reduced anymore. Assume H' has its negative rotation in the outer face f, i.e., without loss of generality the path $\pi_f(t,s)$ belongs to H' and has rotation at most -4. Thus we have again the situation that the two split components H' and H both have a rotation of at most -4 in the outer face. Moreover, these two split components do not contain or overlap each other since s and t are not contained in H' as τ is the new root and H does not contain s' or t' since μ is an ancestor of τ with respect to the old root. Thus we could have reduced the number of bends of H before we changed the root, which is a contradiction to the assumption we made that the number of bends of principal split components with more than three bends

cannot be reduced anymore. Hence, all new principal split components either have at most three bends or they have their negative rotation in some inner face. Finally, we obtain a valid orthogonal representation with at most three bends per principal split component with respect to τ. □

3.4 Optimal Drawings with Fixed Planar Embedding

All results from the previous sections deal with the case where we are only interested in the decision problem of whether a given graph has a valid drawing or not. More precisely, we always assumed to have a valid orthogonal representation of an instance of FlexDraw and showed that this implies that there exists another valid orthogonal representation with certain properties. In this section, we consider positive-convex instances of the optimization problem OptimalFlexDraw. The following generic theorem shows that the results for FlexDraw that we presented so far can be extended to OptimalFlexDraw.

Theorem 3.3. *If the existence of a valid orthogonal representation of an instance of FlexDraw with positive flexibility implies the existence of a valid orthogonal representation with property P, then every positive-convex instance of OptimalFlexDraw has an optimal drawing with property P.*

Proof. Let G be a positive-convex instance of OptimalFlexDraw. Let further $\mathcal{R}$ be an optimal orthogonal representation. We can reinterpret G as an instance of FlexDraw with positive flexibility by setting the flexibility of an edge with ρ bends in $\mathcal{R}$ to $\max\{\rho, 1\}$. Then $\mathcal{R}$ is obviously a valid orthogonal representation of G with respect to these flexibilities. Thus there exists another valid orthogonal representation $\mathcal{R}'$ having property P. It remains to show that $\text{cost}(\mathcal{R}') \leq \text{cost}(\mathcal{R})$ holds when going back to the optimization problem OptimalFlexDraw. However, this is clear for the following reason. Every edge e has as most as many bends in $\mathcal{R}'$ as in $\mathcal{R}$ except for the case where e has one bend in $\mathcal{R}'$ and zero bends in $\mathcal{R}$. In the former case the monotony of $\text{cost}_e(\cdot)$ implies that the cost did not increase. In the latter case e causes the same amount of cost in $\mathcal{R}$ as in $\mathcal{R}'$ since $\text{cost}_e(0) = \text{cost}_e(1) = b_e$ holds for positive-convex instances of OptimalFlexDraw. Note that this proof still works, if the cost functions are only monotone but not convex. □

It follows that every positive-convex 4-planar graph has an optimal drawing that is nice since Theorem 3.3 shows that Theorem 3.2 can be applied. Thus, it is sufficient to consider only nice drawings when searching for an optimal solution, as there exists a nice optimal solution. This is a fact that we crucially exploit in the next section since although the cost function of a principal split component may be non-convex, we can show that it is convex in the interval that is of interest when only considering nice drawings.

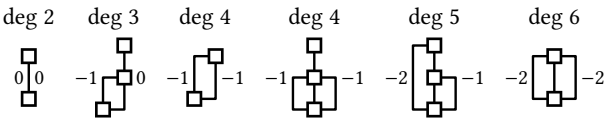


Figure 3.9: Split components with as few bends as possible.

3.5 Optimal Drawings with Variable Planar Embedding

All results we presented so far were based on a fixed planar embedding of the input graph G. In this section we present an algorithm that computes an optimal drawing of G in polynomial time, optimizing over all planar embeddings of G. Our algorithm crucially relies on the existence of a nice drawing among all optimal drawings of G. For biconnected graphs (Section 3.5.1) we present a dynamic program that computes the cost function of all principal split components bottom-up in the SPQR-tree with respect to a chosen root. To compute the optimal drawing among all drawings that are nice with respect to the chosen root, it remains to consider the embeddings of the root itself. If we choose every node to be the root once, this directly yields an optimal drawing of G taking all planar embeddings into account. In Section 3.5.2 we extend our results to connected graphs that are not necessarily biconnected. To this end we first modify the algorithm for biconnected graphs such that it can compute an optimal drawing with the additional requirement that a specific vertex lies on the outer face. Then we can use the BC-tree to solve OptimalFlexDraw for connected graphs. We use the computation of a minimum-cost flow in a network of size n as a subroutine and denote the consumed running time by $T_{\text{flow}}(n)$. In Section 3.5.3 we consider which running time we actually need.

3.5.1 Biconnected Graphs

In this section we always assume G to be a biconnected 4-planar graph forming a positive-convex instance of OptimalFlexDraw. Let $\mathcal{T}$ be the SPQR-tree of G. As defined before, an orthogonal representation is optimal if it has the smallest possible cost. We call an orthogonal representation τ-*optimal* if it has the smallest possible cost among all orthogonal representation that are nice with respect to the root τ. We say that it is $(\tau, \mathcal{E})$-*optimal* if it causes the smallest possible amount of cost among all orthogonal representations that are nice with respect to τ and induce the planar embedding $\mathcal{E}$ on $\text{skel}(\tau)$. In this section we concentrate on finding a $(\tau, \mathcal{E})$-optimal orthogonal representation with respect to a root τ and a given planar embedding $\mathcal{E}$ of $\text{skel}(\tau)$. Then a τ-optimal representation can be computed by choosing every possible embedding of $\text{skel}(\tau)$. An optimal solution can then be computed by choosing every node in $\mathcal{T}$ to be the root once.

In Section 3.3 we extended the terms “bends” and “flexibility”, which were originally defined for single edges, to arbitrary principal split components with respect to the

chosen root. We start out by making precise what we mean with the cost function $\text{cost}_H(\cdot)$ of a principal split component H with poles s and t. Recall that the number of bends of H with respect to an orthogonal representation $\mathcal{S}$ with s and t on the outer face f is defined to be $\max\{|\text{rot}_{\mathcal{S}}(\pi_f(s,t))|, |\text{rot}_{\mathcal{S}}(\pi_f(t,s))|\}$. Assume $\mathcal{S}$ is the nice orthogonal representation of H that has the smallest possible cost among all nice orthogonal representations with ρ bends. Then we essentially define $\text{cost}_H(\rho)$ to be the cost of $\mathcal{S}$. However, with this definition the cost function of H is not defined for all $\rho \in \mathbb{N}_0$ since H does not have an orthogonal representation with zero bends at all, if $\deg(s) > 1$ or $\deg(t) > 1$, as at least one of the paths $\pi_f(s,t)$ and $\pi_f(t,s)$ has negative rotation in this case. More precisely, if $\deg(s) + \deg(t) > 2$, then H has at least one bend, and if $\deg(s) + \deg(t) > 4$, then H has at least two bends. Figure 3.9 shows for each combination of degrees a small example with the smallest possible number of bends. In these two cases we formally set $\text{cost}_H(0) = \text{cost}_H(1)$ and $\text{cost}_H(0), \text{cost}_H(1) = \text{cost}_H(2)$, respectively. Thus, we only need to compute the cost functions for at least $\lceil(\deg(s) + \deg(t) - 2)/2\rceil$ bends. We denote this lower bound by $\ell_H = \lceil(\deg(s) + \deg(t) - 2)/2\rceil$. Hence, it remains to compute the cost function $\text{cost}_H(\rho)$ for $\rho \in [\ell_H, 3]$. For more than three bends we formally set the cost to ∞. Note that the definition of the cost function only considers nice orthogonal representations (including that they are tight). As a result of this restriction the cost for an orthogonal representation with ρ bends might be less than $\text{cost}_H(\rho)$. However, due to Theorem 3.2 in combination with Theorem 3.3 we know that optimizing over nice orthogonal representations is sufficient to find an optimal solution.

As for single edges, we define the *base cost* b_H of the principal split component H to be $\text{cost}_H(0)$. We will see that the cost function $\text{cost}_H(\cdot)$ is monotone and even convex in the interval $[0,3]$ (except for a special case) and thus the base cost is the smallest possible amount of cost that has to be payed for every orthogonal drawing of H. The only exception is the case where $\deg(s) = \deg(t) = 3$. In this case H has at least two bends and thus the cost function $\text{cost}_H(\cdot)$ needs to be considered only on the interval $[2,3]$. However, it may happen that $\text{cost}_H(2) > \text{cost}_H(3)$ holds in this case. Then we set the base cost b_H to $\text{cost}_H(3)$ such that the base cost b_H is really the smallest possible amount of cost that need to be payed for every orthogonal representation of H. We obtain the following theorem.

Theorem 3.4. *If the poles of a principal split component do not both have degree 3, then its cost function is convex on the interval* $[0,3]$.

Before showing Theorem 3.4 we just assume that it holds and moreover we assume that the cost function of every principal split component is already computed. We first show how these cost functions can then be used to compute an optimal drawing. To this end, we define a flow network on the skeleton of the root τ of the SPQR-tree, similar to Tamassias flow network [Tam87]. The cost functions computed for the children of τ will be used as cost functions on arcs in the flow network. As we can

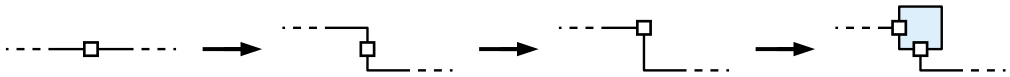


Figure 3.10: A single vertex can be replaced by a split component with three bends.

only solve flow networks with convex costs we somehow have to deal with potentially non-convex cost functions for the case that both endvertices of a virtual edge have degree 3 in its expansion graph. Our strategy is to simply ignore these subgraphs by contracting them into single vertices. Note that the resulting vertices have degree 2 since the poles of graphs with non-convex cost functions have degree 3. The process of replacing the single vertex in the resulting drawing by the contracted component is illustrated in Figure 3.10. The following lemma justifies this strategy.

Lemma 3.4. *Let G be a biconnected positive-convex instance of* OptimalFlexDraw *with τ-optimal orthogonal representation $\mathcal{R}$ and let H be a principal split component with non-convex cost function and base cost b_H. Let further G' be the graph obtained from G by contracting H into a single vertex and let $\mathcal{R}'$ be a τ-optimal orthogonal representation of G'. Then $\text{cost}(\mathcal{R}) = \text{cost}(\mathcal{R}') + b_H$ holds.*

Proof. Assume we have a τ-optimal orthogonal representation $\mathcal{R}$ of G inducing the orthogonal representation $\mathcal{S}$ on H. As H has either two or three bends we can simply contract it yielding an orthogonal representation $\mathcal{R}'$ of G with $\text{cost}(\mathcal{R}') = \text{cost}(\mathcal{R}) - \text{cost}(\mathcal{S}) \leq \text{cost}(\mathcal{R}) - b_H$. The opposite direction is more complicated. Assume we have an orthogonal representation $\mathcal{R}'$ of G', then we want to construct an orthogonal representation $\mathcal{R}$ of G with $\text{cost}(\mathcal{R}) = \text{cost}(\mathcal{R}') + b_H$. Let $\mathcal{S}$ be an orthogonal representation of H causing only b_H cost. Since $\text{cost}_H(\cdot)$ was assumed to be non-convex, $\mathcal{S}$ needs to have three bends. It is easy to see that $\mathcal{R}'$ and $\mathcal{S}$ (or $\mathcal{S}'$ obtained from $\mathcal{S}$ by mirroring the drawing) can be combined to an orthogonal representation of G if the two edges incident to the vertex v in G' corresponding to H have an angle of 90° between them. However, this can always be ensured without increasing the costs of $\mathcal{R}'$. Let e_1 and e_2 be the edges incident to v and assume they have an angle of 180° between them in both faces incident to v. If neither e_1 nor e_2 has a bend, the flex graph contains the cycle around v due to the fact that e_1 and e_2 have positive flexibilities. Bending along this cycles introduces a bend to each of the edges, thus we can assume without loss of generality that e_1 has a bend in $\mathcal{R}'$. Moving v along the edge e_1 until it reaches this bend decreases the number of bends on e_1 by one and ensures that v has an angle of 90° in one of its incident faces. Thus we can replace v by the split component H with orthogonal representation $\mathcal{S}$ having cost b_H yielding an orthogonal representation $\mathcal{R}$ of G with $\text{cost}(\mathcal{R}) = \text{cost}(\mathcal{R}') + b_H$. □

When computing a $(\tau, \mathcal{E})$-optimal orthogonal representation of G we make use of Lemma 3.4 in the following way. If the expansion graph H corresponding to a virtual

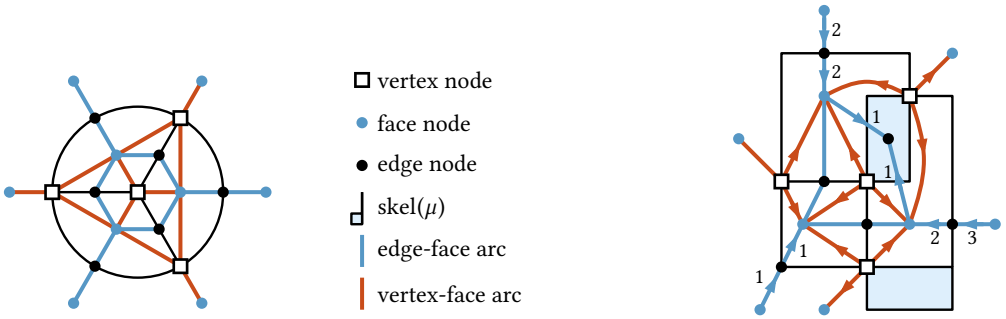


Figure 3.11: (a) The structure of the flow network $N^{\mathcal{E}}$ for the case that τ is an R-node with $\text{skel}(\tau) = K_4$. The outer face is split into several vertices to improve readability. (b) A flow together with the corresponding orthogonal representation. The numbers indicate the amount of flow on the arcs. Undirected edges imply 0 flow, directed arcs without a number have flow 1.

edge ε in $\text{skel}(\tau)$ has a non-convex cost function, we simply contract this virtual edge in $\text{skel}(\tau)$. Note that this is equivalent to contracting H in G. We can then make use of the fact that all remaining expansion graphs have convex cost functions to compute a $(\tau,\mathcal{E})$-optimal orthogonal representation of the resulting graph yielding a $(\tau,\mathcal{E})$-optimal orthogonal representation of the original graph G since the contracted expansion graphs can be inserted due to Lemma 3.4. Note that expansion graphs with non convex cost functions can only appear if the root is a Q- or an S-node. In the skeletons of P- and R-nodes every vertex has degree at least three, thus the poles of an expansion graph cannot have degree 3 since G has maximum degree 4.

Now we are ready to define the flow network $N^{\mathcal{E}}$ on $\text{skel}(\tau)$ with respect to the fixed embedding $\mathcal{E}$ of $\text{skel}(\tau)$; see Figure 3.11a for an example. For each vertex v, each virtual edge ε and each face f in $\text{skel}(\tau)$ the flow network $N^{\mathcal{E}}$ contains the nodes v, ε and f, called *vertex node*, *edge node* and *face node*, respectively. The network $N^{\mathcal{E}}$ contains the arcs (v,f) and (f,v) with capacity 1, called *vertex-face arcs*, if the vertex v and the face f are incident in $\text{skel}(\tau)$. For every virtual edge ε we add *edge-face arcs* (ε,f) and (f,ε), if f is incident to ε. We use $\text{cost}_H(\cdot) - b_H$ as cost function of the arc (f,ε), where H is the expansion graph of the virtual edge ε. The edge-face arcs (ε,f) in the opposite direction have infinite capacity with 0 cost. It remains to define the demand of every node in $N^{\mathcal{E}}$. Every inner face has a demand of 4, the outer face has a demand of -4. An edge node ε stemming from the edge $\varepsilon = \{s,t\}$ with expansion graph H has a demand of $\deg_H(s) + \deg_H(t) - 2$, where $\deg_H(v)$ denotes the degree of v in H. The demand of a vertex node v is $4 - \deg_G(v) - \deg_{\text{skel}(\tau)}(v)$.

In the flow network $N^{\mathcal{E}}$ the flow entering a face node f using a vertex-face arc or an edge-face arc is interpreted as the rotation at the corresponding vertex or along the path between the poles of the corresponding child, respectively; see Figure 3.11b for

an example. Incoming flow is positive rotation and outgoing flow negative rotation. Let $b_{H_1}, \dots, b_{H_k}$ be the base costs of the expansion graphs corresponding to virtual edges in skel(τ). We define the *total base costs* of τ to be $b_\tau = \sum_i b_{H_i}$. Note that the total base costs of τ are a lower bound for the costs that have to be paid for every orthogonal representation of G. We show that an optimal flow ϕ in $N^{\mathcal{E}}$ corresponds to a $(\tau, \mathcal{E})$-optimal orthogonal representation $\mathcal{R}$ of G. Since the base costs do not appear in the flow network, the costs of the flow and its corresponding orthogonal representation differ by the total base costs b_τ, i.e., cost($\mathcal{R}$) = cost(ϕ) + b_τ. We obtain the following lemma.

Lemma 3.5. *Let G be a biconnected positive-convex instance of* OptimalFlexDraw*, let $\mathcal{T}$ be its SPQR-tree with root τ and let $\mathcal{E}$ be an embedding of* skel(τ)*. If the cost function of every principal split component is known, a $(\tau, \mathcal{E})$-optimal solution can be computed in $O(T_{\text{flow}}(|\text{skel}(\tau)|))$ time.*

Proof. As mentioned before, we want to use the flow network $N^{\mathcal{E}}$ to compute an optimal orthogonal representation. To this end we show two directions. First, given a $(\tau, \mathcal{E})$-optimal orthogonal representation $\mathcal{R}$, we obtain a feasible flow ϕ in $N^{\mathcal{E}}$ such that cost(ϕ) = cost($\mathcal{R}$) − b_τ, where b_τ are the total base costs. Conversely, given an optimal flow ϕ in $N^{\mathcal{E}}$, we show how to construct an orthogonal representation $\mathcal{R}$ such that cost($\mathcal{R}$) = cost(ϕ) + b_τ. As the flow network $N^{\mathcal{E}}$ has size $O(|\text{skel}(\tau)|)$, the claimed running time follows immediately.

Let $\mathcal{R}$ be a $(\tau, \mathcal{E})$-optimal orthogonal representation of G. As we only consider nice and thus only tight drawings we can assume the orthogonal representation $\mathcal{R}$ to be tight. Recall that being tight implies that the poles of the expansion graph of every virtual edge have a rotation of 1 in the internal faces. We first show how to assign flow to the arcs in $N^{\mathcal{E}}$. It can then be shown that the resulting flow is feasible and causes cost($\mathcal{R}$) − b_τ cost. For every pair of vertex-face arcs (f, v) and (v, f) in $N^{\mathcal{E}}$ there exists a corresponding face f in the orthogonal representation $\mathcal{R}$ of G and we set $\phi((v, f)) = \text{rot}(v_f)$. Let $\varepsilon = \{s, t\}$ be a virtual edge in skel(μ) incident to the two faces f_1 and f_2. Without loss of generality let $\pi_{f_1}(s, t)$ be the path belonging to the expansion graph of ε. Then $\pi_{f_2}(t, s)$ also belongs to H. We set $\phi((\varepsilon, f_1)) = \text{rot}_{\mathcal{R}}(\pi_{f_1}(s, t))$ and $\phi((\varepsilon, f_2)) = \text{rot}_{\mathcal{R}}(\pi_{f_2}(t, s))$. For the resulting flow ϕ we need to show that the capacity of every arc is respected, that the demand of every vertex is satisfied, and that cost(ϕ) = cost($\mathcal{R}$) − b_τ holds.

First note that the flow on the vertex-face arcs does not exceed the capacities of 1 since every vertex has degree at least 2. Since no other arc has a capacity, it remains to deal with the demands and the costs.

For the demands we consider each vertex type separately. Let f be a face node. The total incoming flow entering f is obviously equal to the rotation in $\mathcal{R}$ around the face f. As $\mathcal{R}$ is an orthogonal representation this rotation equals to 4 (−4 for the outer face), which is exactly the demand of f. Let ε be an edge node corresponding to the expansion

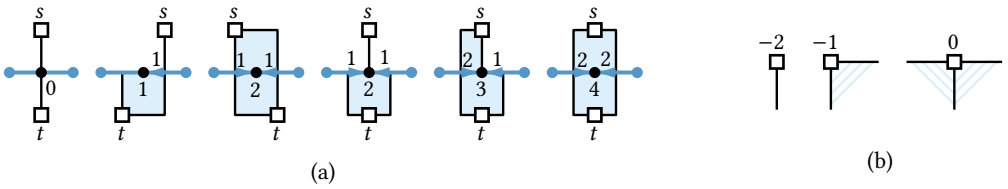


Figure 3.12: (a) Illustration of the demand of virtual edges. (b) Rotation of poles in the outer face, depending on the degree.

graph H with poles s and t. Recall that $\text{dem}(\varepsilon) = \deg_H(s) + \deg_H(t) - 2$ is the demand of ε. Figure 3.12a illustrates the demand of a virtual edge. Let $\mathcal{S}$ be the orthogonal representation induced on H by $\mathcal{R}$ and let f be the outer face of $\mathcal{S}$. Clearly, the flow leaving ε is equal to $\text{rot}_{\mathcal{R}}(\pi_{f_1}(s,t)) + \text{rot}_{\mathcal{R}}(\pi_{f_2}(t,s)) = \text{rot}_{\mathcal{S}}(\pi_f(s,t)) + \text{rot}_{\mathcal{S}}(\pi_f(t,s))$. Since f is the outer face of H, the total rotation around this faces sums up to -4. The rotation of the pole s in the outer face f is $\deg_H(s) - 3$, see Figures 3.12b, and the same holds for t. Thus we have $\text{rot}_{\mathcal{S}}(\pi_f(s,t)) + \text{rot}_{\mathcal{S}}(\pi_f(t,s)) + \deg_H(s) - 3 + \deg_H(t) - 3 = -4$. This yields for the outgoing flow $\text{rot}_{\mathcal{S}}(\pi_f(s,t)) + \text{rot}_{\mathcal{S}}(\pi_f(t,s)) = 2 - \deg_H(s) - \deg_H(t)$, which is exactly the negative demand of ε. It remains to consider the vertex nodes. Let v be a vertex node, recall that $\text{dem}(v) = 4 - \deg_G(v) - \deg_{\text{skel}(\tau)}(v)$ holds. The outgoing flow leaving v is equal to the summed rotation of v in faces not belonging to expansion graphs of virtual edges in $\text{skel}(\tau)$. As $\mathcal{R}$ is an orthogonal representation, the total rotation around every vertex v is $2 \cdot (\deg_G(v) - 2)$. Moreover, v is incident to $\deg_{\text{skel}(\tau)}(v)$ faces that are not contained in expansion graphs of virtual edges of $\text{skel}(\tau)$. Thus there are $\deg_G(v) - \deg_{\text{skel}(\tau)}(v)$ faces incident to v belonging to expansion graphs. As we assumed that the orthogonal representation of every expansion graph is tight, the rotation of v in each of these faces is 1. Thus the rotation of v in the remaining faces not belonging to expansion graphs is $2 \cdot (\deg_G(v) - 2) - (\deg_G(v) - \deg_{\text{skel}(\tau)}(v))$. Rearrangement yields a rotation, and thus an outgoing flow, of $\deg_G(v) + \deg_{\text{skel}(\tau)}(v) - 4$, which is the negative demand of v.

To show that $\text{cost}(\phi) = \text{cost}(\mathcal{R}) - b_\tau$ holds it suffices to consider the flow on the edge-face arcs as no other arcs cause cost. Let ε be a virtual edge and let f_1 and f_2 the two incident faces. The flow entering f_1 or f_2 does not cause any cost, as (ε, f_1) and (ε, f_2) have infinite capacity with 0 cost. Thus only flow entering ε over the arcs (f_1, ε) and (f_2, ε) may cause cost. Assume without loss of generality that the number of bends ρ the expansion graph H of ε has is determined by the rotation along $\pi_{f_1}(s,t)$, i.e., $\rho = -\text{rot}_{\mathcal{R}}(\pi_{f_1}(s,t))$. Let $\rho' = -\text{rot}_{\mathcal{R}}(\pi_{f_2}(t,s))$ be the negative rotation along the path $\pi_{f_2}(t,s)$ in the face f_2. Note that $\phi((f_1,\varepsilon)) = \rho$ and $\phi((f_2,\varepsilon)) = \rho'$. Obviously, the flow on (f_1,ε) causes the cost $\text{cost}_H(\rho) - b_H$. We show that the cost caused by the flow on (f_2,ε) is 0. If $\rho' \leq 0$ this is obviously true, as there is no flow on the edge (f_2,ε). Otherwise, $0 < \rho' \leq \rho$ holds. It follows that the smallest possible number of bends ℓ_H every orthogonal representation of H has lies between ρ' and ρ. It follows

from the definition of $\text{cost}_H(\cdot)$ and from the fact that all cost functions are convex that $\text{cost}_H(\rho') = b_H$. To sum up, the total cost on edge-face arcs incident to the virtual edge ε is equal to the cost caused by its expansion graph H with respect to the orthogonal representation $\mathcal{R}$ minus the base cost b_H. As neither ϕ nor $\mathcal{R}$ have additional cost we obtain $\text{cost}(\phi) = \text{cost}(\mathcal{R}) - b_\tau$.

It remains to show the opposite direction, i.e., given an optimal flow ϕ in $N^{\mathcal{E}}$, we can construct an orthogonal representation $\mathcal{R}$ of G such that $\text{cost}(\mathcal{R}) = \text{cost}(\phi) + b_\tau$. This can be done by reversing the construction above. The flow on edge-face arcs determines the number of bends for the expansion graphs of each virtual edge. The cost functions of these expansion graphs guarantee the existence of orthogonal representations with the desired rotations along the paths between the poles, thus we can assume to have orthogonal representations for all children. We combine these orthogonal representations by setting the rotations between them at common poles as specified by the flow on vertex-face arcs. It can be easily verified that this yields an orthogonal representation of the whole graph G by applying the above computation in the opposite direction. □

The above results rely on the fact that the cost functions of principal split components are convex as stated in Theorem 3.4 and that they can be computed efficiently. In the following we show that Theorem 3.4 really holds with the help of a structural induction over the SPQR-tree. More precisely, the cost functions of principal split components corresponding to the leaves of $\mathcal{T}$ are the cost functions of the edges and thus they are convex. For an inner node μ we assume that the pertinent graphs of the children of μ have convex cost functions and show that $H = \text{pert}(\mu)$ itself also has a convex cost function. The proof is constructive in the sense that it directly yields an algorithm to compute these cost functions bottom up in the SPQR-tree.

Note that we can again apply Lemma 3.4 in the case that the cost function of the expansion graph of one of the virtual edges in $\text{skel}(\mu)$ is not convex due to the fact that both of its poles have degree 3. This means that we can simply contract such a virtual edge (corresponding to a contraction of the expansion graph in H), compute the cost function for the remaining graph instead of H and plug the contracted expansion graph into the resulting orthogonal representations. Thus we can assume that the cost function of each of the expansion graphs is convex, without any exceptions.

The flow network $N^{\mathcal{E}}$ that was introduced to compute an optimal orthogonal representation in the root of the SPQR-tree can be adapted to compute the cost function of the principal split component H corresponding to a non-root node μ. To this end we have to deal with the parent edge, which does not occur in the root of $\mathcal{T}$, and we consider a parameterization of $N^{\mathcal{E}}$ to compute several optimal orthogonal representations with a prescribed number of bends, depending on the parameter in the flow network. Before we describe the changes in the flow network we need to make some considerations about the cost function. By the definition of the cost

function it explicitely optimizes over all planar embeddings of skel(μ). Moreover, as the cost function $\text{cost}_H(\rho)$ depends on the number of bends ρ a graph H has, it implicitly allows to flip the embedding of H since the number of bends is defined as $\max\{|\operatorname{rot}(\pi(s,t))|, |\operatorname{rot}(\pi(t,s))|\}$. However, the flow network $N^{\mathcal{E}}$ can only be used to compute the cost function for a fixed embedding. Thus we define the *partial cost function* $\text{cost}_H^{\mathcal{E}}(\rho)$ of H with respect to the planar embedding $\mathcal{E}$ of skel(μ) to be the smallest possible cost of an orthogonal representation inducing the planar embedding $\mathcal{E}$ on skel(μ) with ρ bends such that the number of bends is determined by $\pi_f(s,t)$, i.e., $\operatorname{rot}(\pi_f(s,t)) = -\rho$, where f is the outer face. Note that the minimum over the partial cost functions $\text{cost}_H^{\mathcal{E}}(\cdot)$ and $\text{cost}_H^{\mathcal{E}'}(\cdot)$, where $\mathcal{E}'$ is obtained by flipping the embedding $\mathcal{E}$ of skel(μ) yields a function describing the costs of H with respect to the embedding $\mathcal{E}$ of skel(μ) depending on the number of bends H has (and not on the rotation along $\pi_f(s,t)$ as the partial cost function does). Obviously, minimizing over all partial cost functions yields the cost function of H.

The flow network $N^{\mathcal{E}}$ is defined as before with the following modifications. The parent edge of skel(μ) does not have a corresponding edge node. Let f_1 and f_2 be the faces in skel(μ) incident to the parent edge. These two faces together form the outer face f of H, thus we could merge them into a single face node. However, not merging them has the advantage that the incoming flow in f_1 and f_2 corresponds to the rotations along $\pi_f(s,t)$ and $\pi_f(t,s)$, respectively (it might be the other way round but we can assume this situation without loss of generality). Thus, we do not merge f_1 and f_2, which enables us to control the number of bends of H by setting the demands of f_1 and f_2. This is also the reason why we remove the vertex-face arcs between the poles and the two faces f_1 and f_2. Before we describe how to set the demands of f_1 and f_2, we fit the demands of the poles to the new situation. As we only consider tight orthogonal representations we know that the rotation at the poles s and t in all inner faces is 1. Thus, we set $\text{dem}(s) = 2 - \deg_{\text{skel}(\mu)}(s)$ and $\text{dem}(t) = 2 - \deg_{\text{skel}(\mu)}(t)$ as this is the number of faces incident to s and t, respectively, after removing the vertex-face arcs to f_1 and f_2. With these modifications the only flow entering f_1 and f_2 comes from the paths $\pi_f(s,t)$ and $\pi_f(t,s)$, respectively. As the total rotation around the outer face is -4 and the rotation at the vertices s and t is $\deg_H(s)-3$ and $\deg_H(t)-3$, respectively, we have to ensure that $\text{dem}(f_1) + \text{dem}(f_2) = 2 - \deg_H(s) - \deg_H(t)$. As mentioned before, we assume without loss of generality that $\pi_f(s,t)$ belongs to the face f_1 and $\pi_f(t,s)$ belongs to f_2. Then the incoming flow entering f_1 corresponds to $\operatorname{rot}(\pi_f(s,t))$ of an orthogonal representation. We parameterize $N^{\mathcal{E}}$ with respect to the faces f_1 and f_2 starting with $\text{dem}(f_1) = 0$ and $\text{dem}(f_2) = 2 - \deg_H(s) - \deg_H(t)$. It obviously follows that an optimal flow in $N^{\mathcal{E}}$ with respect to the parameter ρ corresponds to an optimal orthogonal representation of H that induces $\mathcal{E}$ on skel(μ) and has a rotation of $-\rho$ along $\pi_f(s,t)$. Thus, up to the total base costs b_μ, the cost function of the flow network equals to the partial cost function of H on the interval $[\ell_H, 3]$, i.e.,

$\text{cost}_{N^\varepsilon}(\rho) + b_\mu = \text{cost}_H^{\mathcal{E}}(\rho)$ for $\ell_H \leq \rho \leq 3$. To obtain the following lemma it remains to show two things for the case that $\deg(s) + \deg(t) < 6$. First, $\text{cost}_{N^\varepsilon}(\rho)$ and thus each partial cost function is convex for $\ell_H \leq \rho \leq 3$. Second, the minimum over these partial cost functions is convex.

Lemma 3.6. *If Theorem 3.4 holds for each principal split component corresponding to a child of the node μ in the SPQR-tree, then it also holds for* $\text{pert}(\mu)$.

Proof. As mentioned before, we can use the flow network $N^{\mathcal{E}}$ to compute the partial cost function $\text{cost}_H^{\mathcal{E}}(\rho)$ for $\ell_H \leq \rho \leq 3$ since $\text{cost}_H^{\mathcal{E}}(\rho) = \text{cost}_{N^\varepsilon}(\rho) + b_\mu$ holds on this interval. In the following we only consider the case where $\deg_H(s) + \deg_H(t) < 6$ holds for the poles s and t. For the case $\deg_H(s) = \deg_H(t) = 3$ we do not need to show anything. To show that the partial cost function is convex we do the following. First, we show that $\text{cost}_H^{\mathcal{E}}(\rho)$ is minimal for $\rho = \ell_H$. This implies that the cost function $\text{cost}_{N^\varepsilon}(\rho)$ of the flow network is minimal for $\rho = \rho_0 \leq \ell_H$. Then Theorem 1.1 can be applied showing that $\text{cost}_{N^\varepsilon}(\rho)$ is convex for $\rho \in [\rho_0, \infty]$ yielding that the partial cost function $\text{cost}_H^{\mathcal{E}}(\rho)$ is convex for $\rho \in [\ell_H, 3]$. Thus, it remains to show that $\text{cost}_H^{\mathcal{E}}(\rho)$ is minimal for $\rho = \ell_H$ to obtain convexity for the partial cost functions.

Let $\mathcal{S}$ be an orthogonal representation of H with $\rho \in [\ell_H, 3]$ bends such that $\pi_f(s,t)$ determines the number of bends, i.e., $\text{rot}_{\mathcal{S}}(\pi_f(s,t)) = -\rho$, where f is the outer face of H. We show the existence of an orthogonal representation S' with $\text{rot}_{\mathcal{S}'}(\pi_f(s,t)) = -\ell_H$ and $\text{cost}(S') \leq \text{cost}(S)$. Since we assume $\mathcal{S}$ to be tight, the rotations at the poles $\text{rot}_{\mathcal{S}}(s_f)$ and $\text{rot}_{\mathcal{S}}(t_f)$ only depend on the degree of s and t. More precisely, we have $\text{rot}_{\mathcal{S}}(s_f) = \deg_H(s) - 3$ and the same holds for t. Since the total rotation around the outer face f is -4 the following equation holds.

$$\text{rot}_{\mathcal{S}}(\pi_f(t,s)) = \rho + 2 - \deg_H(s) - \deg_H(t) \tag{3.1}$$

In the following we show that $\text{rot}_{\mathcal{S}}(\pi_f(t,s)) \geq 0$ holds if the number of bends ρ exceeds ℓ_H. Then Corollary 3.1 in combination with Theorem 3.3 can be used to reduce the rotation along $\pi_f(t,s)$ and thus reduce the number of bends by 1, yielding finally an orthogonal representation with ℓ_H bends determined by $\pi_f(s,t)$. Recall that the lower bound for the number of bends was defined as $\ell_H = \lceil(\deg(s) + \deg(t) - 2)/2\rceil$. First consider the case that $\deg_H(s) + \deg_H(t)$ is even (and of course less than 6). Then Equation (3.1) yields $\text{rot}_{\mathcal{S}}(\pi_f(t,s)) = \rho - 2\ell_H$. If ρ is greater than ℓ_H this yields $\text{rot}_{\mathcal{S}}(\pi_f(t,s)) > -\ell_H$. Since ℓ_H is at most 1 in the case that $\deg(s) + \deg(t)$ is even and less than 6, this yields $\text{rot}_{\mathcal{S}}(\pi_f(t,s)) > -1$. The case that $\deg_H(s) + \deg_H(t)$ is odd works similarly. Then Equation (3.1) yields $\text{rot}_{\mathcal{S}}(\pi_f(t,s)) = \rho - 2\ell_H + 1$. As before ρ is assumed to be greater than ℓ_H yielding $\text{rot}_{\mathcal{S}}(\pi_f(t,s)) > -\ell_H + 1$. As ℓ_H is at most 2 we again obtain $\text{rot}_{\mathcal{S}}(\pi_f(t,s)) > -1$, which concludes the proof that the partial cost functions are convex.

It remains to show that the minimum over the partial cost functions is convex. First assume that μ is an R-node. Then its skeleton has only two embeddings $\mathcal{E}$ and $\mathcal{E}'$

where $\mathcal{E}'$ is obtained by flipping $\mathcal{E}$. We have to show that the minimum over the two partial cost functions $\text{cost}_H^{\mathcal{E}}(\cdot)$ and $\text{cost}_H^{\mathcal{E}'}(\cdot)$ remains convex. For the case that $\deg(s)+\deg(t) = 5$ the equation $\ell_H = 2$ holds and thus we only have to show convexity on the interval $[2,3]$. Obviously, $\text{cost}_H(\cdot)$ is convex on this interval if and only if $\text{cost}_H(2) \leq \text{cost}_H(3)$. As this is the case for both partial cost functions, it is also true for the minimum. For $\deg(s) + \deg(t) < 5$ we first show that $\text{cost}_H^{\mathcal{E}}(\ell_H) = \text{cost}_H^{\mathcal{E}'}(\ell_H)$ holds. For the case that $\deg(s)+\deg(t)$ is even this is clear since mirroring an orthogonal representation $\mathcal{S}$ with $\text{rot}_{\mathcal{S}}(\pi_f(s,t)) = -\ell_H$ inducing $\mathcal{E}$ on $\text{skel}(\mu)$ yields an orthogonal representation $\mathcal{S}'$ with $\text{rot}_{\mathcal{S}'}(\pi_f(s,t)) = -\ell_H$ inducing $\mathcal{E}'$ on $\text{skel}(\mu)$. For the case that $\deg(s) + \deg(t) = 3$, the orthogonal representation $\mathcal{S}$ with rotation -1 along $\pi_f(s,t)$ can also be mirrored yielding $\mathcal{S}'$ with rotation 0 along $\pi_f(s,t)$. By Corollary 3.1 this rotation can be reduced to -1 without causing any additional cost. As this construction also works in the opposite direction we have $\text{cost}_H^{\mathcal{E}}(\ell_H) = \text{cost}_H^{\mathcal{E}'}(\ell_H)$ for all cases. Moreover, $\text{cost}_H^{\mathcal{E}}(0) = \text{cost}_H^{\mathcal{E}}(1)$ holds by definition, if $\deg(s) + \deg(t) > 2$. If $\deg(s) = \deg(t) = 1$ this equation is also true as the rotation along $\pi_f(s,t)$ of an orthogonal representation can be reduced by 1 if it is 0, again due to Corollary 3.1. Thus it remains to show that the cost function $\text{cost}_H(\cdot)$ defined as the minimum of $\text{cost}_H^{\mathcal{E}}(\cdot)$ and $\text{cost}_H^{\mathcal{E}'}(\cdot)$ is convex on the interval $[1,3]$.

Assume for a contradiction that $\text{cost}_H(\rho)$ is not convex for $\rho \in [1,3]$, that is, $\Delta\,\text{cost}_H(1) > \Delta\,\text{cost}_H(2)$. Assume without loss of generality that $\text{cost}_H(3) = \text{cost}_H^{\mathcal{E}}(3)$ holds. As we showed before $\text{cost}_H(1) = \text{cost}_H^{\mathcal{E}}(1)$ also holds. Since $\text{cost}_H(2)$ is the minimum over $\text{cost}_H^{\mathcal{E}}(2)$ and $\text{cost}_H^{\mathcal{E}}(2)$ we additionally have $\text{cost}_H(2) \leq \text{cost}_H^{\mathcal{E}}(2)$. This implies that the inequalities $\Delta\,\text{cost}_H^{\mathcal{E}}(1) \geq \Delta\,\text{cost}_H(1)$ and $\Delta\,\text{cost}_H^{\mathcal{E}}(2) \leq \Delta\,\text{cost}_H(2)$ hold, yielding that the partial cost function $\text{cost}_H^{\mathcal{E}}(\rho)$ is not convex for $\rho \in [1,3]$, which is a contradiction. Thus $\text{cost}_H(\cdot)$ is convex.

The case that μ is a P-node works similar to the case that μ is an R-node. If μ has only two children, its skeleton has only two embeddings $\mathcal{E}$ and $\mathcal{E}'$ obtained from one another by flipping. Thus the same argument as for R-nodes applies. If μ has three children, then $\deg(s) = \deg(t) = 3$ holds and thus we do not have to show convexity. Note that in the case $\deg(s) = \deg(t) = 3$ the resulting cost function can be computed by taking the minimum over the partial cost functions with respect to all embeddings of $\text{skel}(\mu)$, although it may by non-convex. If μ is an S-node, we have a unique embedding and thus the partial cost function with respect to this embedding is already the cost function of H. Note that considering only the rotation along $\pi_f(s,t)$ for the partial cost function is not a restriction, as S-nodes are completely symmetric. □

Lemma 3.6 together with the fact that the cost function of every edge is convex shows that Theorem 3.4 holds, i.e., the cost functions of all principal split components are convex on the interesting interval $[0,3]$ except for the special case where both poles have degree 3. However, this special case is easy to handle as principal split components of this type with non-convex cost functions can be simply contracted

to a single vertex by Lemma 3.4. Moreover, the proof is constructive in the sense that it shows how the cost functions can be computed efficiently bottom up in the SPQR-tree. For each node μ we have to solve a constant number of minimum-cost flow problems in a flow network of size $O(|\operatorname{skel}(\mu)|)$. As the total size of all skeletons in $\mathcal{T}$ is linear in the number n of vertices in G, we obtain an overall $O(T_{\text{flow}}(n))$ running time to compute the cost functions with respect to the root τ. Finally, Lemma 3.5 can be applied to compute an optimal orthogonal representation with respect to a fixed root and a fixed embedding of the root's skeleton in $O(T_{\text{flow}}(|\operatorname{skel}(\tau)|))$ time. To compute an overall optimal solution, we have to compute a $(\tau, \mathcal{E})$-optimal solution for every root τ and every embedding $\mathcal{E}$ of $\operatorname{skel}(\tau)$. The number of embeddings of $\operatorname{skel}(\tau)$ is linear in the size of $\operatorname{skel}(\tau)$ (since P-nodes have at most degree 4) and the total size of all skeletons is linear in n. We obtain the following theorem.

Theorem 3.5. *OptimalFlexDraw can be solved in $O(n \cdot T_{\text{flow}}(n))$ time for positive-convex biconnected instances.*

3.5.2 Connected Graphs

In this section we extend the result obtained in Section 3.5.1 to the case that the input graph G contains cutvertices. Let $\mathcal{B}$ be the BC-tree of G rooted at some B-node β. Then every Block except for β has a unique cutvertex as parent and we need to find optimal orthogonal representations with the restriction that this cutvertex lies on the outer face. We claim that we can then combine these orthogonal representations of the blocks without additional cost.

Unfortunately, with the so far presented results we cannot compute the optimal orthogonal representation of a biconnected graph considering only embeddings where a specific vertex v lies on the outer face. We may restrict the embeddings of the skeletons we consider when traversing the SPQR-tree bottom up to those who have v on the outer face. However, we can then no longer assume that the cost functions we obtain are symmetric. To deal with this problem, we present a modification of the SPQR-tree, that can be used to represent exactly the planar embeddings that have v on the outer face and are represented by the SPQR-tree rooted at a node τ.

Let τ be the root of the SPQR-tree $\mathcal{T}$. If v is a vertex of $\operatorname{skel}(\tau)$, then restricting the embeddings of $\operatorname{skel}(\tau)$ to those who have v on the outer face of $\operatorname{skel}(\tau)$ forces v to be on the outer face of the resulting embedding of G. Otherwise, v is contained in the expansion graph of a unique virtual edge ε in $\operatorname{skel}(\tau)$, we say that v is *contained* in ε. Obviously, ε has to be on the outer face of the embedding of $\operatorname{skel}(\tau)$. However, this is not sufficient and it depends on the child μ of τ corresponding to ε whether v lies on the outer face of the resulting embedding of G. Let $\mathcal{E}_\tau$ be an embedding of $\operatorname{skel}(\tau)$ having ε on the outer face and let s and t be the endpoints of ε. Then there are two possibilities, either $\varepsilon = \{s, t\}$ has the outer face to the left or to the right, where the

terms “left” and “right” are with respect to an orientation from t to s. Assume without loss of generality that the outer face lies to the right of ε and consider the child μ of τ corresponding to ε. As $\mathcal{T}$ is rooted, we consider only embeddings of skel(μ) that have the parent edge $\{s,t\}$ on the outer face. As the choice of the outer face of skel(μ) does not have any effect on the resulting embedding, we can assume that $\{s,t\}$ lies to the left of skel(μ), i.e., the inner face incident to $\{s,t\}$ lies to the right of $\{s,t\}$ with respect to an orientation from t to s. A vertex contained in skel(μ) then lies obviously on the outer face of the resulting embedding of G if and only if it lies on the outer face of the embedding of skel(μ). Thus, if v is contained in skel(μ), restricting the embedding choices such that v lies on the outer face of skel(μ) forces v to be on the outer face of G. Note that in this case μ is either an R- or an S-node. For S-nodes there is no embedding choice and every vertex in skel(μ) lies on the outer face in this embedding. If μ is an R-node, there are only two embeddings and either v lies on the outer face of exactly one of them or in none of them. In the latter case the SPQR-tree with respect to the root τ does not represent an embedding of G with v on the outer face at all.

Assume that v is not contained in skel(μ). Then it is again contained in a single virtual edge ε' and it is necessary that ε' lies on the outer face of the embedding of skel(μ). Moreover, it depends on the child of μ corresponding to ε' whether v really lies on the outer face. Note that fixing ε' on the outer face completely determines the embedding of skel(μ) if it is not a P-node. If μ is a P-node, the virtual edge ε' has to be the rightmost, whereas the order of all other virtual edges can be chosen arbitrarily. If this is the case we split the P-node into two parts, one representing the fixed embedding of ε', the other representing the choices for the remaining edges; see Figure 3.13a. More precisely, we split μ into two P-nodes, the first one containing the parent edge $\{s,t\}$, the edge ε' and a new virtual edge corresponding to the second P-node, which is inserted as child. The skeleton of the second P-node contains a parent edge corresponding to the first P-node and the remaining virtual edges that were contained in skel(μ) but are not contained in the first P-node. The children of μ are attached to the two P-nodes depending on where the corresponding virtual edges are. Note that by splitting the P-node μ, the virtual edge ε' can no longer be in between two other virtual edges in μ. However, this is a required restriction, thus we do not loose embeddings that we want to represent. Moreover, the new P-node containing the virtual edge ε' that need to be fixed to the outer face contains only two virtual edges (plus the parent edge) and thus the embedding of its skeleton is completely fixed by requiring ε' to be on the outer face.

To sum up, if skel(τ) contains v, then we simply have to choose an embedding of skel(τ) with v on the outer face. Otherwise, we have to fix the virtual edge containing v to the outer face and additionally have to consider the child of τ corresponding to this virtual edge. For the child we then have essentially the same situation. Either v is contained in its skeleton, then the embedding is fixed to the unique embedding

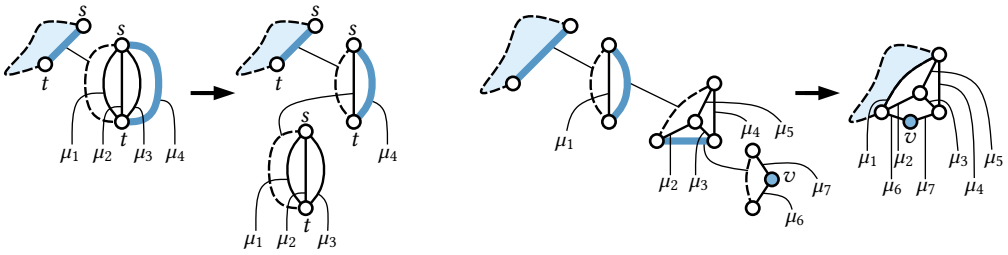


Figure 3.13: (a) Splitting a P-node into two P-nodes, the vertex v fixed to the outer face is contained in the blue bold edges. (b) Contracting the path from the root to the node containing v in its skeleton.

having v on the outer face or v is contained in some virtual edge. However, then the embedding of the skeleton is again completely fixed (P-nodes have to be split up first) and we can continue with the child corresponding to the virtual edge containing v. This yields a path of nodes starting with the root τ having a completely fixed embedding only depending on the embedding $\mathcal{E}_\tau$ chosen for skel(τ). As the nodes on the path do not represent any embedding choices, we can simply contract the whole path into a single new root node, merging the skeletons on the path, such that the embedding of the new skeleton of the root is still fixed. This contraction is illustrated in Figure 3.13b. More precisely, let τ be the root and let ε be the edge containing v, corresponding to the child μ. Then we merge τ and μ by replacing ε in τ by the skeleton of μ without the parent edge. The children of μ are of course attached to the new root τ' since skel(τ') contains the corresponding virtual edges. As mentioned before, the embedding of skel(μ) was fixed by the requirement that v is on the outer face, thus the new skeleton skel(τ') has a unique embedding $\mathcal{E}_{\tau'}$ inducing $\mathcal{E}_\tau$ on skel(τ) and having v or the new virtual edge containing v on the outer face. The procedure of merging the root with the child corresponding to the virtual edge containing v is repeated until v is contained in the skeleton of the root. We call the resulting tree the *restricted SPQR-tree* with respect to the vertex v and to the embedding $\mathcal{E}_\tau$ of the root.

To come back to the problem OptimalFlexDraw, we can easily apply the algorithm presented in Section 3.5.1 to the restricted SPQR-tree. All nodes apart from the root are still S-, P-, Q- or R-nodes and thus the cost functions with respect to the corresponding pertinent graphs can be computed bottom up. The root τ may have a more complicated skeleton, however, its embedding is fixed, thus we can apply the flow algorithm as before, yielding an optimal drawing with respect to the chosen root τ and to the embedding $\mathcal{E}_\tau$ of skel(τ) with the additional requirement that v lies on the outer face. Since the *restricted SPQR-tree* can be easily computed in linear time for a chosen root τ and a fixed embedding $\mathcal{E}$ of skel(τ), we can compute a $(\tau, \mathcal{E})$-optimal orthogonal

representation with the additional requirement that v lies on the outer face in $T_{\text{flow}}(n)$ time, yielding the following theorem.

Theorem 3.6. *OptimalFlexDraw with the additional requirement that a specific vertex lies on the outer face can be solved in $O(n \cdot T_{\text{flow}}(n))$ time for positive-convex biconnected instances.*

As motivated before, we can use the BC-tree to solve OptimalFlexDraw for instances that are not necessarily biconnected. We obtain the following theorem.

Theorem 3.7. *OptimalFlexDraw can be solved in $O(n^2 \cdot T_{\text{flow}}(n))$ time for positive-convex instances.*

Proof. Let G be a positive-convex instance with positive flexibility of OptimalFlexDraw and let $\mathcal{B}$ be its BC-tree rooted at some B-node β. We show how to find an optimal drawing of G, optimizing over all embeddings represented by $\mathcal{B}$ with respect to the root β. Then we can simply choose every B-node in $\mathcal{B}$ to be the root once, solving OptimalFlexDraw. The algorithm consumes $O(n \cdot T_{\text{flow}}(n))$ time for each root β and thus the overall running time is $O(n^2 \cdot T_{\text{flow}}(n))$. For the block corresponding to the root β we use Theorem 3.5 to find the optimal orthogonal representation. For all other blocks we use Theorem 3.6 to find the optimal orthogonal representation with the cutvertex corresponding to the parent in $\mathcal{B}$ on the outer face. It remains to stack these orthogonal representations together without causing additional cost. This can be easily done, if a cutvertex that is forced to lie on the outer face has all free incidences in the outer face and every other cutvertex has all free incidences in a single face. The former can be achieved as we can assume orthogonal representations to be tight. If the latter condition is violated by a cutvertex v, then v has two incident edges e_1 and e_2 and the rotation of v is 0 in both incident faces. If both edges e_1 and e_2 have zero bends, we bend along a cycle around v in the flex graph and thus we can assume without loss of generality that e_1 has a bend. Moving v along e_1 to this bend yields an orthogonal representation where v has both free incidences in the same face. Thus given the orthogonal representations for the blocks, we can simply stack them together without causing additional cost. □

3.5.3 Computing the Flow

In the previous sections we used $T_{\text{flow}}(n)$ as placeholder for the time necessary to compute a minimum-cost flow in a flow network of size n. Most minimum-cost flow algorithms do not consider the case of multiple sinks and sources. However, this is not a real problem as we can simply add a *supersink* connected to all sinks and a *supersource* connected to all sources. Unfortunately, the resulting flow network is no longer planar. Orlin [Orl93] gives a strongly polynomial time minimum-cost flow algorithm with running time $O(m \log n(m + n \log n))$, where n is the number of vertices and m the

number of arcs. Since our flow network is planar (plus supersink and supersource) the number of arcs is linear in the number of nodes. Thus with this flow algorithm we have $T_{\text{flow}}(n) \in O(n^2 \log^2 n)$.

This can be slightly improved using the algorithm by Borradaile et al. [Bor+11] to compute a feasible flow in a planar flow network with multiple sources and sinks, consuming $O(n \log^3 n)$ time. Afterwards, it remains to minimize the cost in the residual network. As this network is planar, the shortest path computation in the algorithm by Orlin [Orl93] can be done in linear time due to Henzinger et al. [Hen+97], yielding the running time $O(n^2 \log n)$.

Cornelsen and Karrenbauer give a minimum-cost flow algorithm for planar flow networks with multiple sources and sinks consuming $O(\sqrt{\chi}\, n \log^3 n)$ time [CK12], where χ is the cost of the resulting flow. Since the cost functions in an instance of OptimalFlexDraw may define exponentially large costs in the size of the input, we cannot use this flow algorithm in general to obtain a polynomial time algorithm. However, in practice it does not really make sense to have exponentially large costs. Moreover, in several interesting special cases, an optimal solution has cost linear in the number of vertices. We obtain the following results.

Corollary 3.3. *A positive-convex instance G of OptimalFlexDraw can be solved in $O(n^4 \log n)$ or $O(\sqrt{\chi}\, n^3 \log^3 n)$ time, where χ is the cost of an optimal solution. The running time can be improved by a factor of $O(n)$ for biconnected graphs.*

3.6 Conclusion

We have presented an efficient algorithm for the problem OptimalFlexDraw. As a first step, we have considered biconnected 4-planar graphs with a fixed embedding and have shown that they always admit a nice drawing, which implies at most three bends per edge except for a single edge on the outer face with up to four bends.

Our algorithm for optimizing over all planar embeddings requires that the first bend on every edge does not cause any cost as the problem becomes NP-hard otherwise. Apart from that restriction we allow the user to specify an arbitrary convex cost function independently for each edge. This enables the user to control the resulting drawing.

In particular, our algorithm can be used to minimize the total number of bends, neglecting the first bend of each edge. This special case is the natural optimization problem arising from the decision problem FlexDraw. As another interesting special case, one can require every edge to have at most two bends and minimize the number of edges having more than one bend. This enhances the algorithm by Biedl and Kant [BK98] generating drawings with at most two bends per edge with the possibility of optimization. Note that in both special cases the cost of an optimal solution is linear

in the size of the graph, yielding a running time in $O(n^{\frac{7}{2}} \log^3 n)$ ($O(n^{\frac{5}{2}} \log^3 n)$ if the graph is biconnected).

We want to conclude with some notes on the open question already mentioned in the conclusion of Chapter 2. There we asked whether one can obtain an FPT-algorithm for OptimalFlexDraw with respect to the number of inflexible edges. I.e., we allow k edges to already cause cost with the first bend and want to obtain an algorithm that is polynomial in the size of the graph but maybe exponential (or even super-exponential) in k.

To adapt the idea from Chapter 2, one would need to show that the cost function one obtains for a split component in the dynamic program is non-convex only if the split component contains inflexible edges. For each node, one could then split the non-convex cost functions into convex pieces and try all combinations. Unfortunately, this does not work for the following reason. The cost functions of the split components may already be non-convex, even though they do not contain inflexible edges. The algorithm for OptimalFlexDraw presented in this chapter only works due to the fact that the cost functions need to be considered only on a small interval, on which they are convex. However, in the presence of inflexible edges, an optimal drawing may require a split component to have more than three bends even if this particular split component does not contain inflexible edges. Thus, although the algorithms from this and from the previous chapter use similar techniques, the algorithm for OptimalFlexDraw does not allow even a small number of inflexible edges.

4 Higher-Degree Nodes in the Kandinsky Model

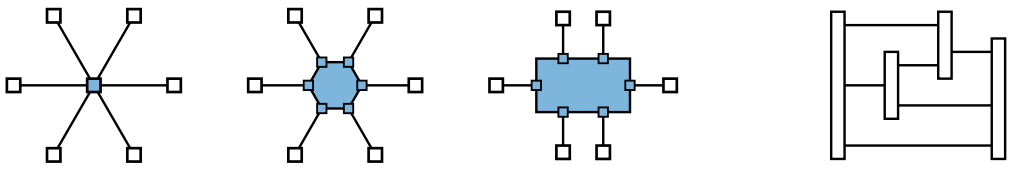
In this chapter, we consider the Kandinsky model for orthogonal drawings that allows vertices of degree larger than 4. We show that finding bend-minimal Kandinsky drawings of plane graphs is NP-complete, which solves a long-standing open problem. On the positive side, we give an efficient algorithm for several restricted variants, such as graphs of bounded branch width and a subexponential exact algorithm for general plane graphs.

This chapter is based on joint work with Guido Brückner and Ignaz Rutter [BBR14].

4.1 Introduction

The main drawback of the previous chapters, and of orthogonal drawings in general, is the restriction to graphs with maximum degree 4. The usual solution to this issue is to represent the vertices by rectangles instead of points, which allows multiple edges to leave a vertex on the same side. The first algorithm for minimizing the number of bends in such a drawing was given by Batini et al. [BTT84]. Their idea was to replace each vertex of degree greater than 4 with a cycle. Then one can apply the bend-minimization algorithm for 4-planar graphs by Tamassia [Tam87], additionally enforcing that the cycles representing high-degree vertices are drawn as rectangles; see Figure 4.1a. This approach is usually referred to as the *Giotto model*.

The general issue with the Giotto model is that the rectangles representing vertices may be very large. In fact, every planar graph admits a *visibility representation* where every vertex is represented by a vertical bar and every edge is represented by a horizontal line segment; see Figure 4.1b. Although a visibility representation is technically an orthogonal drawing (with very high and thin rectangles) without any bends, it is



(a) (b)

Figure 4.1: (a) In the Giotto model, a high-degree vertex is replaced with a cycle that is forced to have rectangular shape. (b) A visibility representation of the complete graph on four vertices.

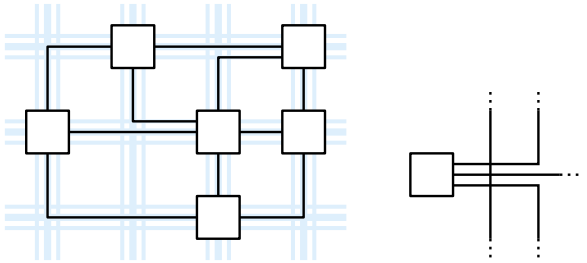
surely not what the user usually expects. Thus, algorithms using the Giotto model usually add some restrictions. In the original algorithm by Batini et al. [BTT84], the edges incident to a vertex are required to be evenly distributed over the four sides of the vertex, e.g., if a vertex has degree 8, then two edges enter the vertex on each of the sides top, bottom, left, and right. This approach was reconsidered by Tamassia et al. [TDB88], who coined the term Giotto. With this additional requirement, it can still happen that vertices are very large. However, for practical instances, the vertices are usually not too large.

Another way to distribute the edges incident to a vertex to the four sides was given by Batini et al. [BNT86]. They compute a so-called k-gonal representation with the minimum number of bends. The concept of k-gonal representations is a direct extension of orthogonal representations to more than 4 directions. In contrast to orthogonal representations, the resulting k-gonal representation has in general no geometric realization. However, the slope of an edge in the k-gonal representation can be used to determine the side at which it leaves its incident vertex. This way of distributing the edges seems more natural than just distributing them evenly to the four sides.

Another way of restricting the Giotto model is to require vertices of maximum degree 4 to have only a single edge in each direction [KM98]. This restriction makes the most sense if many vertices have degree 4 or less. If all vertices have larger degree, the algorithm may actually compute a visibility representation, as mentioned before. This approach was proposed in conjunction with the quasi-orthogonal drawing style, where a post-processing step replaces each rectangle representing a high-degree vertex with a single point inside the rectangle. The edge segments incident to these points may then be non-orthogonal. This post-processing step could of course be also applied when there are other restrictions (e.g., the ones mentioned above).

To summarize, the Giotto model allows drawing arbitrary plane graphs orthogonally with the minimum number of bends. The drawback of too large vertices is reduced by requiring additional restriction that usually lead to smaller vertices and more bends. However, the user has no influence on how large specific vertices are and there is no guarantee that the vertices are not too large. To the best of our knowledge, there exists unfortunately no paper comparing the effect of different types of restrictions to the size of the vertices.

To overcome the issue of generating potentially large vertices, Fößmeier and Kaufmann [FK95] defined the *Kandinsky model* (originally called podevsnef). In a Kandinsky drawing, vertices are mapped to squares of constant size centered at grid points on a coarse grid, while edges are routed on a finer grid; see Figure 4.2a and also Section 1.4.5. This allows several edges to emanate from the same side of a vertex while the vertices have constant size. Similar to the work by Tamassia [Tam87] on orthogonal drawings, Fößmeier and Kaufmann transformed the problem Kandinsky Bend Minimization



(a) (b)

Figure 4.2: (a) A Kandinsky drawing of the wheel of size 5. (b) An edge crossing three edges close to a vertex. This is not allowed in a Kandinsky drawing when representing the crossings with vertices.

for plane graphs into the problem of computing a minimum-cost flow in a network. The resulting flow network has *bundle capacities*, i.e., the total amount of flow on some pairs of edges is limited. It has been claimed that this flow network can be reduced to an ordinary minimum-cost flow network, which can then be solved efficiently [FK95]. Unfortunately, this reduction turned out to be flawed [Eig03]. Before showing in this chapter that bend minimization in the Kandinsky model is actually NP-hard, we mention several papers on the Kandinsky model. They are mostly ordered chronologically.

Fößmeier et al. [FKK97] showed that every plane graph admits a 1-bend Kandinsky drawing that can be computed in linear time. They also adapt the original bend-minimization algorithm to compute bend minimum Kandinsky drawings under the restriction that every edge has at most one bend. As for the original algorithm, the reduction to a minimum-cost flow network is flawed. In fact, our NP-hardness proof also shows hardness for this restricted version of bend minimization.

Another extension of the Kandinsky model aims for drawing planarizations of non-planar graphs. To do that, one could simply handle the dummy vertices (representing crossings) as normal vertices. However, this may require more bends than actually necessary; see Figure 4.2b. Fößmeier and Kaufmann [FK97] extend the Kandinsky model to treat dummy vertices in a special way such that these unnecessary bends are avoided. The bend-minimization problem in this setting can again be reduced to an NP-hard flow network with bundle capacities. Fößmeier and Kaufmann also run experiments for this model using an ILP solver, which performs well.

Bertolazzi et al. introduce the *simple-Kandinsky model* [BDD00]. It requires that all bends that correspond to a $0°$ angle at a vertex must be right bends. This leads to drawings with more bends than in the general Kandinsky model. However, it eliminates the bundle capacities in the flow network, making bend minimization in this restricted model efficiently solvable. Moreover, they give a branch-and-bound algorithm that optimizes over all planar embeddings of a biconnected graph.

The Kandinsky model was further extended by Di Battista et al. [Di +99] to represent vertices by rectangles of prescribed size (instead of squares of constant size). Their algorithm computing drawings in this extended model relies on an initial drawing

in the basic Kandinsky model. In their experiments they use the simple-Kandinsky model (allowing efficient bend minimization) for the initial drawings.

Eiglsperger et al. [EFK00] give an ILP-formulation computing bend-minimum Kandinsky drawings subject to additional constraints. Among other restrictions, their algorithm also supports vertices of prescribed size.

Brandes et al. [Bra+02] present an algorithm that generates Kandinsky drawings that look similar to a sketch of the graph given in the input. The aim is to find a good trad-off between a drawing with few bends and a drawing that looks similar to the given sketch. The formal problem statement includes the problem Kandinsky Bend Minimization and the algorithm by Brandes et al. [Bra+02] relies on the fact that Kandinsky Bend Minimization can be solved efficiently, which we show to be false unless P = NP. However, their experiments perform well using an ILP formulation.

Eiglsperger points out that the reduction from the flow problem with bundle capacities to a normal flow problem is flawed [Eig03]. He moreover gives a 2-approximation for Kandinsky Bend Minimization. A 2-approximation performing better in practice is given by Barth et al. [BMY07].

Besides these mostly theoretical papers, there are several algorithms generating Kandinsky drawings that are actually of practical use. E.g., algorithms that allow labeled nodes and edges [Bin+05] or algorithms generating layouts of UML class diagrams [Eig+04]. Although the Kandinsky model has received much attention and despite its relevance for practical use, the fundamental question about the complexity of Kandinsky Bend Minimization has remained open for almost two decades.

Contribution and Outline

In this work, we show that Kandinsky Bend Minimization is NP-complete even for graphs with a fixed planar embedding (no matter if we allow or forbid empty faces; see Section 1.4.5). This also holds if each edge may have at most one bend; see Section 4.2. As an intermediate step, we show NP-hardness of the problem Orthogonal 01-Embeddability, which asks whether a plane graph (with maximum degree 4) admits an orthogonal drawing when requiring some edges to have exactly one and the remaining edges to have zero bends. This is an interesting result on its own, as it can serve as tool to show hardness of other orthogonal drawing problems. In particular, it gives a simpler proof for the hardness of deciding whether a 4-planar graph with variable planar embedding has a 0-bend orthogonal drawing.

We then study the complexity of Kandinsky Bend Minimization subject to structural graph parameters in Section 4.3. For graphs with branch width k, we obtain an algorithm with running time $2^{O(k \log n)}$. For fixed branch width this yields a polynomial-time algorithm (running time $O(n^3)$ for series-parallel graphs), for general plane graphs the result is an exact algorithm with subexponential running time $2^{O(\sqrt{n} \log n)}$.

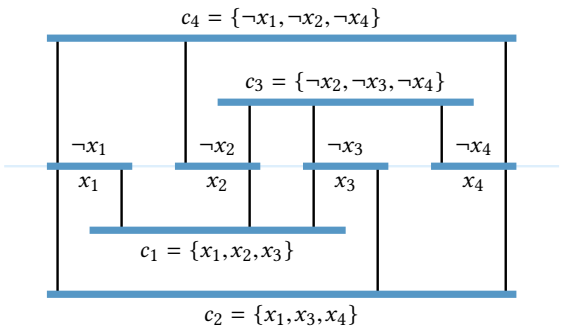


Figure 4.3: The instance of Planar Monotone 3-Sat with variables $x_1, \ldots, x_4$ and clauses $\{x_1, x_2, x_3\}$, $\{x_1, x_3, x_4\}$, $\{\neg x_2, \neg x_3, \neg x_4\}$, and $\{\neg x_1, \neg x_2, \neg x_4\}$.

4.2 Complexity

Let $\mathcal{S} = (\mathcal{X}, C)$ be an instance of 3-Sat with variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$. A clause is a *positive clause* if it contains only positive literals, a *negative clause* if it contains only negative literals, and a *mixed clause* otherwise. In the *variable-clause graph*, every variable and ever clause is a vertex and there is an edge xc connecting a variable $x \in \mathcal{X}$ with a clause $c \in C$ if and only if $x \in c$ or $\neg x \in c$.

In a *monotone rectilinear representation* of the variable-clause graph, the variables are represented as horizontal line segments on the x-axis, the positive and negative clauses are represented as horizontal line segments below and above the x-axis, respectively, and a variable is connected to an adjacent clause by a vertical line segment such that no two line segments cross. Note that an instance admitting a monotone rectilinear representation cannot contain mixed clauses. An instance of Planar Monotone 3-Sat is an instance $S = (\mathcal{X}, C)$ of 3-Sat together with a monotone rectilinear representation of its variable-clause graph; see Figure 4.3 for an example. De Berg and Khosravi [BK12] show that Planar Monotone 3-Sat is NP-hard.

The problem Orthogonal 01-Embeddability is defined as follows. Given a 4-plane graph $G = (V, E)$ and partitioned edge set $E = E_0 \cup E_1$, test whether G admits an orthogonal drawing such that every edge in E_i has exactly i bends. We also refer to the edges in E_0 and E_1 as 0- and 1-edges, respectively. In the following, we always consider the variant of Orthogonal 01-Embeddability where we allow to *fix angles at vertices*, i.e., the value of $\text{rot}_f(v)$ for a vertex v with incident face f might be given with the input. Fixing the angles at vertices does not make the problem harder since augmenting a vertex v to have degree 4 by adding degree-1 vertices incident to v has the same effect as fixing the angles at v (when choosing the planar embedding appropriately). Note that this reduces the case with fixed angles at vertices to the one without fixed angles. In the following we implicitly allow angles at vertices to be fixed.

In this section we first show that Orthogonal 01-Embeddability is NP-hard by a reduction from Planar Monotone 3-Sat. Afterwards, we show that Kandinsky Bend Minimization is NP-hard by a reduction from Orthogonal 01-Embeddability.

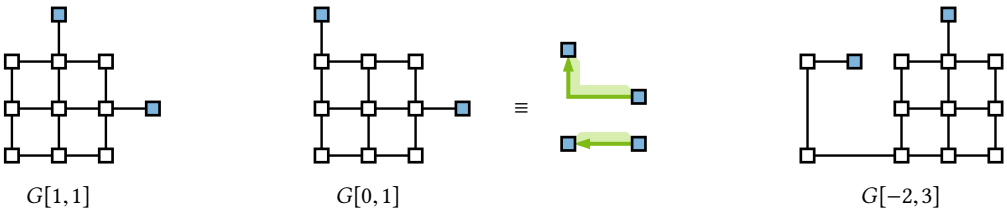


Figure 4.4: Three different interval gadgets. The vertices s and t are marked blue. In future figures, $G[0,1]$ is represented by a directed green edge. It either has no bend or one bend to the right.

4.2.1 Orthogonal 01-Embeddability

Consider a single 1-edge e. When drawing it, we have to make the decision to either bend it in one or the other direction. In the reduction from Planar Monotone 3-Sat, this basic decision will encode the decision to set a variable either to `true` or to `false`. In addition to that, the construction consists of several building blocks. For every variable, we need a gadget that outputs its positive and its negative literal. Moreover, we build gadgets representing clauses that admit a correct drawing if and only if at least one out of three edges that require one bend is bent in the desired direction. Since the same literal usually occurs in several clauses, we need to copy the decision made for one edge to several edges. Finally, we need to bring the decisions of the variables to the clauses without restricting the possible drawings of the clauses too much.

In the following we first present some simple gadgets that are used as building blocks in the following constructions. Then we start with the variable gadget that outputs the positive and negative literal of a variable. Afterwards, we show how to duplicate literals and then present the so called bendable pipes that are used to bring the value of a literal to the clauses. Finally, we present the clause gadget. In the end, we put these building blocks together and show the correctness of the construction.

Building Blocks

An interval gadget is a small graph $G[\rho_1, \rho_2]$ with two designated degree-1 vertices (its *endpoints*) s and t on the outer face. It has the property that the rotation of $\pi(s,t)$ is in the interval $[\rho_1, \rho_2]$ for any orthogonal drawing. The construction is similar to the tendrils used by Garg and Tamassia [GT01]; see Figure 4.4 for some examples.

Lemma 4.1. *The interval gadget $G[\rho_1, \rho_2]$ admits an orthogonal 0-bend drawing with rotation ρ if and only if $\rho \in [\rho_1, \rho_2]$.*

The interval gadget we use most frequently in the following is $G[0,1]$, which behaves like an edge that may have one bend, but only into a fixed direction (recall that the

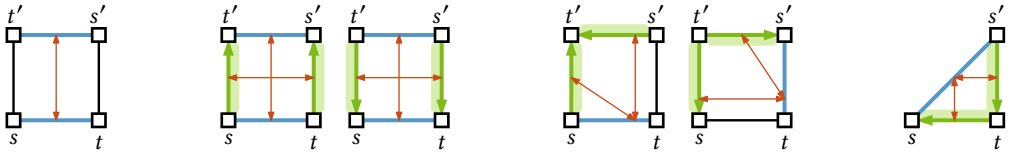


Figure 4.5: Building blocks for our gadgets. The edges are color-coded; 0-edges are black, 1-edges are blue and 01-edges are green and directed such that they may bend right but not left. The building blocks are (a) the box; (b) the bendable box; (c) the merger; (d) the splitter.

planar embedding of our graph is fixed). To simplify the illustrations, we draw $G[0,1]$ as shown in Figure 4.4 and we refer to them as 01-edges.

To simplify the description of the hardness proof, we next describe a number of basic building blocks, which we combine in different ways to obtain the gadgets for our construction. The building blocks are shown in Figure 4.5.

Except for the last of the building blocks, each of them consists of a 4-cycle s, t, s', t'. They only differ in the types of edges. In the *box* st and $s't'$ are 1-edges and the other edges are 0-edges; see Figure 4.5a. In a *bendable box* the two zero-bend edges of a box are replaced by 01-edges directed from t to s' and from t' to s, respectively; see Figure 4.5b. In a *merger* the edge st is a 1-edge, $s't'$ and st' are 01-edges (with this orientation) and ts' is a 0-edge; see Figure 4.5c. Finally, a *splitter* is a 3-cycle s, t, s', where ss' is a 1-edge and $s't$ and ts are 01-edges (with this orientation); see Figure 4.5d.

Symmetric versions of the bendable box and the splitter can be obtained by reversing the directions of both 01-edges, as shown in Figure 4.5b,c. Since they differ from the original only by exchanging the inner and outer face and mirroring the instance, their behavior is completely symmetric. Note that, apart from the 0-edges, all edges of the building blocks admit precisely two possible rotation values in each face. Thus, each edge attains its maximum rotation value in one of its incident faces and the minimum rotation in the other one. We call an orthogonal 01-representation of a building block *right-angled* if all inner angles at vertices are 90°. The following lemma states the functionality of these building blocks, which is essentially that in a right-angled orthogonal 01-representation the rotation values of some of the edges are not independent of one another but are linked in the sense that exactly one of them must attain its minimum (maximum) rotation value in f. In Figure 4.5 such dependencies are displayed as red arrows. We will later interpret the rotation values as an encoding of truth values. The red arrows then correspond to a transmission of the encoded information.

Lemma 4.2. *Consider a building block B and assume that we are given rotation values for each of the edges incident to the inner face f of B that respect the bend constraints of the edges. The following conditions for each of the building blocks are necessary and*

sufficient for the existence of a right-angled orthogonal 01*-representation of* B *respecting the given rotation values.*

(1) *Box: Exactly one of* $\{st, s't'\}$ *attains its minimum (maximum) rotation in* f.

(2) *Bendable box: Exactly one of* $\{st, s't'\}$ *and exactly one of* $\{st', ts'\}$ *attains its minimum (maximum) rotation in* f.

(3) *Merger:* st *attains its minimum (maximum) rotation in* f *if and only if* st' *and* $s't'$ *attain their maximum (minimum) rotation in* f.

(4) *Splitter:* st *attains its minimum (maximum) rotation in* f *if and only if* $s't$ *and* $t's$ *attain their maximum (minimum) rotation in* f.

Proof. We first treat the building blocks that consist of a 4-cycle. Denote the rotation values of $ts, st', t's'$ and $s't$ by $\rho_1, \rho_2, \rho_3, \rho_4$, respectively. Note that, in any valid drawing, each of the vertices contributes a rotation of 1 to the inner face f. Since the total rotation around f must be 4, this implies $\rho_1 + \rho_2 + \rho_3 + \rho_4 = 0$ is necessary and sufficient for the existence of a valid drawing.

For the box, we have $\rho_2 = \rho_4 = 0$, and thus $\rho_1 = -\rho_3$ is necessary and sufficient, which implies the claim.

For the bendable box, observe that $\rho_2 \in \{0,1\}$ and $\rho_4 \in \{-1,0\}$, and thus $\rho_2 + \rho_4 \in \{-1,0,1\}$. Similarly, $\rho_1, \rho_3 \in \{-1,1\}$, and thus $\rho_1 + \rho_3 \in \{-2,0,2\}$. To achieve a total sum of 0, it follows that $\rho_1 + \rho_3 = 0$ and $\rho_2 + \rho_4 = 0$ is necessary and sufficient. The claim follows.

For the merger observe that $\rho_4 = 0$. Moreover, we have $\rho_2 \in \{0,1\}$ and $\rho_3 \in \{-1,0\}$, and thus $\rho_2 + \rho_3 \in \{-1,0,1\}$. Since $\rho_1 \in \{-1,1\}$, it follows that $\rho_2 + \rho_3 = 0$ can be excluded. This together with the fact that $\rho_1 = -\rho_2 - \rho_3$ is necessary and sufficient proves the claim.

Finally, we consider the splitter. We denote the rotations of ss', $s't$ and ts in f by ρ_1, ρ_2 and ρ_3, respectively. Since each of the three vertices incident to f supplies a rotation of 1, the existence of a valid drawing is equivalent to $\rho_1 + \rho_2 + \rho_3 = 1$. Note that $\rho_1 \in \{-1,1\}$, whereas $\rho_2, \rho_3 \in \{0,1\}$, and thus $\rho_2 + \rho_3 \in \{0,1,2\}$. It follows immediately that $\rho_2 + \rho_3 = 1$ is not possible, and thus $\rho_2 = \rho_3$ is necessary. Then $\rho_1 = 1 - 2\rho_2$ follows, showing the claim. □

We will now construct our gadgets from these building blocks. To this end, we take copies of building blocks and glue them together by identifying certain edges (together with their endpoints). As mentioned above, we will use rotations of the 1-edges to encode certain information. Thus, our gadgets will always have such edges on the boundary of the outer face. In the figures, we will again indicate the necessary conditions from Lemma 4.2 by red edges as in Figure 4.5. It follows from Lemma 4.2 that when there is a path of such red edges from one edge to another edge, then they

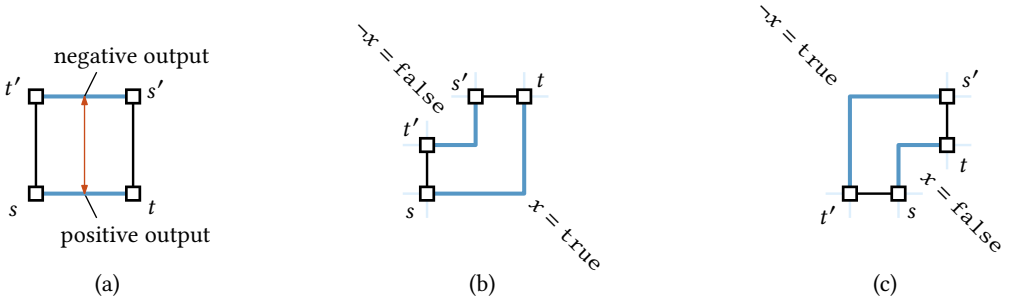


Figure 4.6: (a) The variable gadget. (b–c) The two possible orthogonal representations corresponding to $x = \texttt{true}$ and $x = \texttt{false}$, respectively.

are synchronized. In particular, if both are incident to the outer face than exactly one of them attains the minimum and one of them attains the maximum rotation there in any valid drawing.

Gadget Constructions

Variable Gadget. The variable gadget for a variable x consists of a single box with vertices s, t, s', t'. The two 1-bend edges st and $s't'$ are called the *positive* and *negative output*, respectively. It immediately follows from Lemma 4.2 that it has exactly two different valid drawings. We use the interpretation that x has value `true` if the rotation of the the positive output in the outer face is maximum, and false otherwise; see Figure 4.6. The following lemma summarizes the properties; it follows immediately from Lemma 4.2.

Lemma 4.3. *Assume the rotations ρ_p and ρ_n of the positive and negative output edges in the outer face are fixed. There is a right-angled orthogonal* 01*-representations of the variable gadget respecting ρ_p and ρ_n if and only if $\rho_p = -\rho_n \in \{-1, 1\}$.*

Literal Duplicator. A duplicator is a structure that has three 1-bend edges on the outer face, one of which is the *input edge*, the other two are the *output edges*. The key property is that the structure is such that the state of the inputs is transferred to both outputs in any right-angled orthogonal 01-representation, i.e., the input attains its maximum (minimum) rotation in the outer face if and only if the outputs attains their minimum (maximum) rotation in the outer face. The duplicator is formed by a splitter, which is glued to two mergers via its $\{0, 1\}$-edges; see Figure 4.7. The fact that indeed the information encoded in the input edge is copied to the output edges follows from the red paths connecting the input to the outputs and Lemma 4.2.

Lemma 4.4. *Assume the rotations ρ_i of the input edge and the rotations ρ_o and ρ'_o of the two output edges in the outer face are fixed. There is a right-angled orthogonal* 01*-*

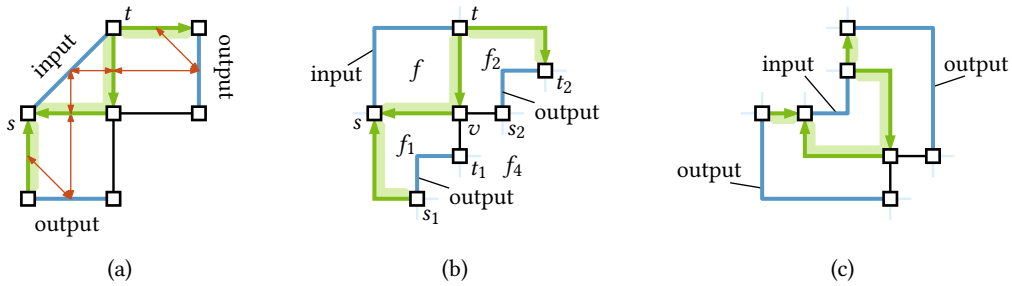


Figure 4.7: (a) The literal duplicator. (b–c) The two possible orthogonal representations corresponding to the values `false` and `true`, respectively.

representations of the variable gadget respecting these rotations if and only if $\rho_i = -\rho_o = -\rho'_o \in \{-1, 1\}$.

By concatenating several duplicators in a tree-like fashion, we can of course take as many copies of the state of a literal as there are clauses containing that literal. We make this more precise later.

Bendable Pipes. The bendable pipe gadget is used for transmitting the information about a literal to a clause. It has an input and an output edge, and has the property that in any valid drawing the information encoded in the input is transmitted to the output. To remedy the fact that the duplicators change their shape depending on the state of the literal they copy, we allow some flexibility of the pipes, allowing them to change how strongly the pipe is bent. This is achieved as follows.

A *zig-zag* consists of a bendable box and a bendable box where the 01-edges are reversed, such that two of their 1-edges are identified. One of the 1-bend edges on the outer face is the input, the other is the output; see Figure 4.8. It follows immediately from Lemma 4.2 that the information from the input is transferred to the output. Moreover, it also follows from Lemma 4.2 that the decision which of the bendable boxes bend their 01-edges can be taken independently. Thus, the zig-zag allows to choose the rotation ρ, ρ' of the paths between the input and the output edge with $\rho = -\rho'$ for each $\rho \in \{-1, 0, 1\}$.

A *k-bendable pipe* is obtained by concatenating k zig-zags; see Figure 4.8e. Again Lemma 4.2 easily implies that the information is transmitted from the input to the output, and moreover, by concatenating suitable drawings of the zig-zags, for each rotation $\rho \in \{-k, \dots, k\}$, the paths between the input and the output edge along the outer face can have rotation ρ and $-\rho$, respectively. In a high-level view, a k-bendable pipe looks like an edge that transfers information between its endpoints and can be bent up to k times either to the left or to the right. The following lemma summarizes the properties of k-bendable pipes.

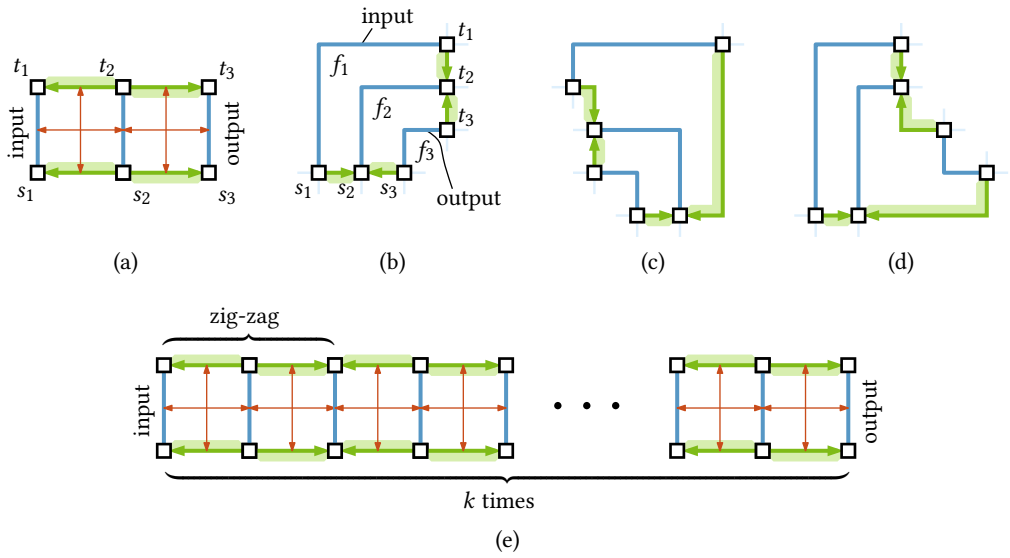


Figure 4.8: (a) The zig-zag. (b–d) Drawings of the zig-zag with different rotations 0, 1, and −1 when the input edge has rotation −1 in the outer face. Corresponding drawings where the rotation of the input edge is +1 are symmetric. (e) The k-bendable pipe.

Lemma 4.5. *Assume the rotations ρ_i and ρ_o of the input edge and the output edge as well as the rotations ρ and ρ' of the two counterclockwise paths on the outer face connecting the input and the output edge are fixed.*

There is a right-angled orthogonal 01*-representations of the k-bendable pipe if and only if $\rho_i = -\rho_o \in \{-1, 1\}$ and $\rho = -\rho' \in \{-k, \dots, k\}$.*

Clause Gadget. The *clause gadget* is a cycle C of length 4, consisting of three 1-edges, the *input edges*, and the interval gadget $G[-2,3]$; see Figure 4.9a. The embedding is fixed such that the inner face of the clause lies to the right of the interval gadget $G[-2,3]$ (i.e., the rotation of $G[-2,3]$ in the inner face lies in the interval $[-2,3]$). Again we only consider right-angled drawings, where the rotations at the vertices in the internal face are all fixed to 1.

The clause gadget interprets a rotation of −1 for an input edge in the inner face as `true` and a rotation of 1 as `false`. In Figure 4.9a all three input edges are set to `true`. In Figure 4.9b two of the three input edges represent the value `false`. In Figure 4.9c all input edges are `false`, thus $G[-2,3]$ would need to have a rotation of −3 in the inner face, which is not possible. The following lemma states more precisely that the clause gadget admits a valid drawing with the given rotations of the input edges if and only if at least one of the input edges represents the value `true`.

Lemma 4.6. *Assume that the rotation ρ_1, ρ_2, ρ_3 of the input edges in the inner face*

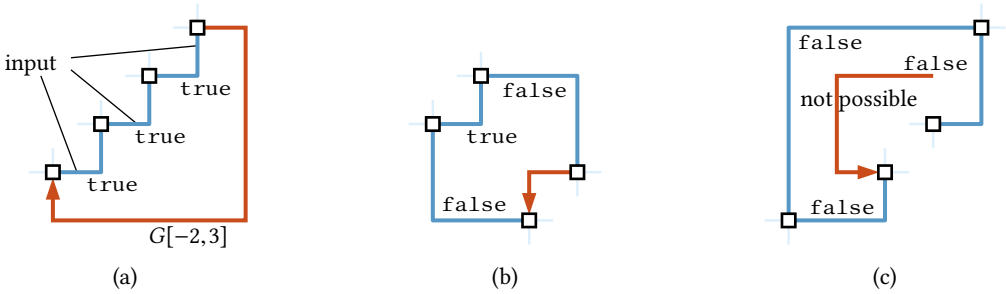


Figure 4.9: (a–c) The clause gadget with three different combinations of values on the input edges.

are fixed. There exists a right-angled orthogonal 01-representation of the clause gadget respecting these rotations if and only if $\rho_i \in \{-1,1\}$ *for* $i \in \{1,2,3\}$ *and* $\rho_i = -1$ *for at least one* $i \in \{1,2,3\}$.

Proof. Each of the four vertices of the clause gadget C has rotation 1 in the inner face of C. Thus the sum of the rotations ρ_1, ρ_2, ρ_3, and the rotation of $G[-2,3]$ in the inner face of C must be 0. The possible rotation of $G[-2,3]$ are exactly the integers in the interval $[-2,3]$ (Lemma 4.1). Thus, we get an orthogonal 01-representation if and only if $\rho_1 + \rho_2 + \rho_3 \in [-3,2]$, which is the case if and only if not all three rotations are 1. □

Putting Things Together

Let $S = (X, C)$ together with a monotone rectilinear representation be an instance of Planar Monotone 3-Sat. The plan is to create a variable gadget for every variable and a clause gadget for every clause, duplicate the literals (using the literal duplicator) outputted by the variable gadget as many times as they occur in clauses, and bring the values of the duplicated literals to the input of the clauses using bendable pipes.

Thus, if we have two gadgets A and B, we want to use an output edge of A as the input edge of B. To make the description simpler, we assume each input edge and each output edge of the gadgets to be oriented such that the outer face lies to its left and to its right, respectively. We can combine A and B by identifying an output edge e_A of A with an input edge e_B of B such that their sources and targets coincide. All input and output edges of the two gadgets remain input and output edges in the resulting graph, except for e_A and e_B.

Let $x \in X$ be a variable. We take one variable gadget X representing the decision made for x. Let k be the number of clauses containing the literal x. We successively add $k-1$ literal duplicators. The input edge of the first literal duplicator is identified with the positive output edge of X. The input edge of every following literal duplicator is identified with an output edge of a previously added literal duplicator. The graph we get has the negative output edge at X and k output edges belonging to literal

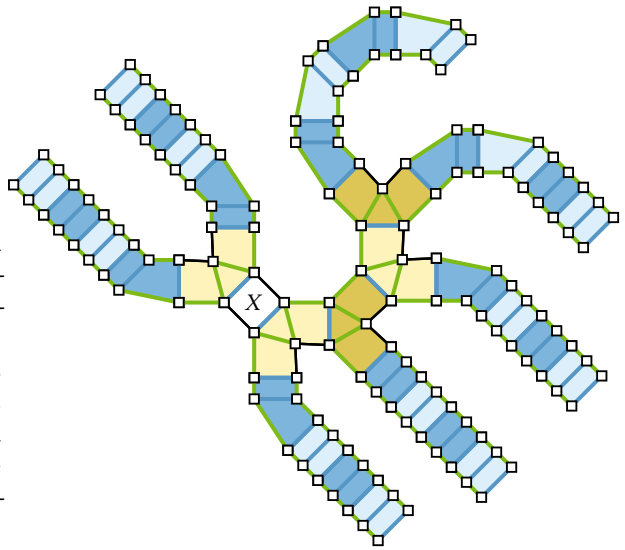


Figure 4.10: Variable tree T_x of a variable x whose positive and negative literal have five and two occurrences, respectively. The variable gadget is shaded white, duplicators are shaded in yellow and zig-zags (forming bendable pipes) are shaded blue. Adjacent gadgets of the same type are shaded with different saturations.

duplicators. To each of these k output edges we add a K-bendable pipe for a suitably large K by identifying the output edge with the input edge of the bendable pipe. We choose $K = 3m^2 + 4m$, where m is the number of edges in the variable-clause graph of S. Let k' be the number of clauses containing the literal $\neg x$. As for the positive literal, we add $k' - 1$ literal duplicators, this time identifying the input edge of the first literal duplicator with the negative output edge of X. As before, we also add K-bendable pipes to each of the k' output edges. We call the resulting graph *variable tree of* x and denote it by T_x. We call the k output edges of the bendable pipes attached to literal duplicators attached to the positive output edge of the variable gadget X the *positive output edges* of T_x. The k' other output edges are *negative output edges* of T_x. The variable tree for the case $k = 5$ and $k' = 2$ is illustrated in Figure 4.10.

For the instance $S = (X, C)$ of Planar Monotone 3-Sat we create the following instance of Orthogonal 01-Embeddability. For every variable $x \in X$, we take the variable tree T_x. For every clause $c \in C$, we add a copy of the clause gadget. We connect them by identifying the output edges of the variable trees with the input edges of the clause gadget in the following way.

Consider a variable x and a positive clause c with $x \in c$ in the monotone rectilinear representation of S. We say that c is the *ith positive clause* of x if the edge connecting c and x is the ith edge incident to x (ordered from left to right). Analogously, x is the *jth variable* of c if this edge is the jth edge incident to c. In the instance shown in Figure 4.3 and Figure 4.11, the clause c_1 is the first positive clause of x_2 and x_2 is the second variable of c_1. Analogously, we define the *ith negative clause*.

Let c be the ith positive clause of x and let x be the jth variable of c. Let further C be the clause gadget corresponding to c. Traversing the outer face of C in counter-

clockwise order starting with the interval gadget defines an order on the input edges of C. Moreover, traversing the variable tree T_x in counter-clockwise order starting with an edge incident to the variable gadget defines an order on the positive output edges of T_x. We identify the ith positive output edge of T_x with the jth input edge of C. For a negative clause containing $\neg x$, we do exactly the same except for defining the order of the negative output edges by traversing the outer face of T_x in clockwise order. This identification of input with output edges is done for every edge in the variable-clause graph. We denote the resulting graph by $G(S)$. Figure 4.11 shows the monotone rectilinear representation (rotated by 45°) of an example instance S and the graph $G(S)$. The graph $G(S)$ has two kinds of faces. Faces that are inner faces in the variable tree or in the clause gadget are called *small faces*. The other faces are *large faces*. Note that there is a one-to-one correspondence between the large faces of $G(S)$ and the faces of the variable-clause graph of S. We obtain the following theorem by proving that S admits a satisfying truth assignment if and only if $G(S)$ admits an orthogonal 01-representation.

Theorem 4.1. *Orthogonal 01-Embeddability is NP-complete.*

Proof. Let $S = (X, C)$ be an instance of Monotone Planar 3-Sat and let $G(S)$ be the graph constructed from S as defined above. We first show that the existence of an orthogonal 01-representation of $G(S)$ implies the existence of a satisfying truth assignment for S.

Let O be an orthogonal 01-representation of $G(S)$. Let $x \in X$ be a variable and let X be the corresponding variable gadget in $G(S)$. If the positive output edge of X has rotation -1 in its outer face, we set $x = \texttt{true}$ (as illustrated in Figure 4.6b). Otherwise, we set $x = \texttt{false}$ (as illustrated in Figure 4.6c). We claim that this gives a satisfying truth assignment for S. Let $c \in C$ be a positive clause and let C be the corresponding clause gadget in $G(S)$. By Lemma 4.6, at least one of the input edges of C has rotation -1 in its inner face. By construction of $G(S)$, this input edge is identified with an positive output edge of the variable tree T_x for a variable x. Let X be the corresponding variable gadget. As there is a path of literal duplicators and bendable pipes from the positive output edge of X to every positive output edge of T_x, it follows from Lemma 4.4 and Lemma 4.5 that the positive output edge of X has rotation -1 in its outer face if and only if any positive output edge of T_x has rotation -1 in the outer face of T_x. Thus, it follows that the positive output edge of X has rotation -1 in its outer face and thus $x = \texttt{true}$, which satisfies the clause c.

If c is a negative clause, we find a variable x such that the negative output edge of the corresponding variable gadget X has rotation -1 in its outer face. By Lemma 4.3, the positive output edge of X has rotation 1 in its outer face, thus $x = \texttt{false}$ holds, which satisfies the negative clause c containing $\neg x$.

It remains to show the opposite direction. Assume we have a satisfying truth assignment for S. We show how to construct an orthogonal 01-representation of $G(S)$.

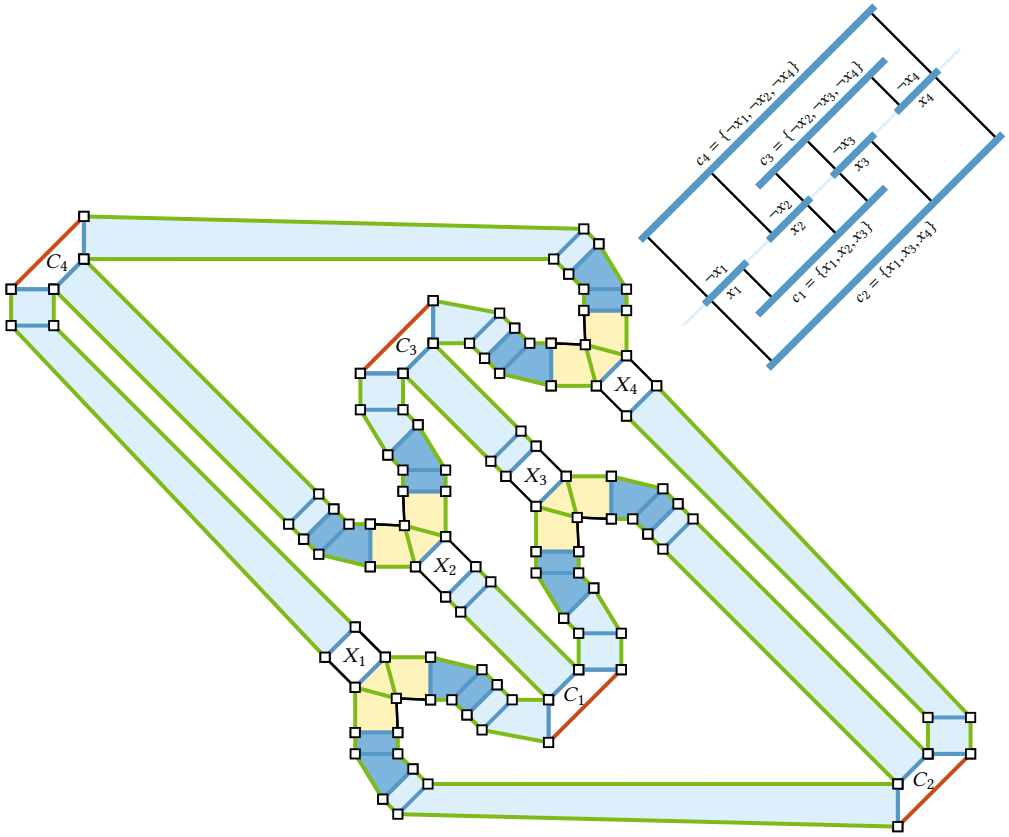


Figure 4.11: Example reduction of Planar Monotone 3-Sat to Orthogonal 01-Embeddability. The bendable pipes have been shortened for clarity.

As $G(S)$ consists of gadgets for which the rotations around every vertex are fixed, it remains to specify a rotation for every edge such that the rotation around every inner face is 4. We start with the small faces. Consider the variable tree T_x of a variable x containing the variable gadget X. If $x = \texttt{true}$, we choose the orthogonal 01-representation of X where the positive output edge has rotation -1. This yields a feasible representation by Lemma 4.3; see Figure 4.6b–c.

This already fixes the rotation of the literal duplicators in T_x that are directly attached to the output edges of X. By Lemma 4.4 this fixes the rotation of the corresponding output edge (to the same behavior as the input edge) and a corresponding orthogonal 01-representation of the duplicator exists; see Figure 4.7b–c. Applying this procedure iteratively to every literal duplicator whose input edge has a fixed rotation fixes the orthogonal representation of every literal duplicator in T_x.

Similarly, we (partially) fix the orthogonal 01-representation of the bendable pipes contained in T_x iteratively according to Lemma 4.5. More precisely, the rotation of the

1-edges is fixed according to the rotation of the input edge; see Figure 4.8e. However, we do not fix the rotation of the bendable pipes. Recall that, by Lemma 4.5 this rotation can be anything in $\{-K, \dots, K\}$. We will need the flexibility of choosing this rotation to get the rotations in the large faces right.

Note that the resulting orthogonal 01-representations of the variable tree have the following properties. The positive output edges of T_x have rotation -1 if $x = \texttt{true}$ and rotation 1 otherwise. The negative output edges have rotation -1 if $\neg x = \texttt{true}$ and rotation 1 otherwise. By fixing the orthogonal representations of the variable trees in this way, we already fix the orthogonal representation of the input edges of the clause gadgets in $G(S)$. Let C be a clause gadget in $G(S)$. Since S is a satisfying truth assignment, it follows that the rotation of at least one input edge of C in the inner face of C is -1. Thus, C admits an orthogonal 01-representation by Lemma 4.6. The choices made so far imply that every small face in our orthogonal 01-representation has rotation 4, as required.

It remains to choose the rotations of the bendable pipes such that the rotation in the large inner faces is 4. Initially, assume that the rotation of every bendable pipe is 0. We first bound the maximum deviation from a rotation of 4 around large faces.

Let f be a large face and let f_S be the corresponding face in the variable-clause graph of S. The boundary of f can be naturally subdivided into paths belonging to different variable trees and paths on the outer face of clause gadgets. Let x be a variable on the boundary of f_S. A path between two output edges of T_x consists of three subpaths. Two paths with rotation 0 consisting of edges belonging to bendable pipes and, in between, one path of edges belonging to literal duplicators. Clearly, this path has length at most $\deg(x)$ and since the absolute value of the rotation at edges and vertices is at most 1, we get a total rotation between $-2\deg(x)$ and $2\deg(x)$ in the large face. Summing over all variables incident to f_S gives us a rotation between $-2m$ and $2m$, where m is the number of edges in the variable-clause graph. Moreover, for each clause incident to f_S the boundary of f contains a path having absolute rotation at most 3. As there are $m/3$ clauses, the total rotation around the large face f is between $-3m$ and $3m$.

Changing the rotation of a bendable pipe increases the rotation of one incident large face by 1 and decreases it in the other incident large face by -1. (Note that this does not affect the rotations at small faces.) Thus, choosing the rotations of the bendable pipes such that the rotation in every large face is 4 (except for the outer face with rotation -4) is equivalent to finding a flow in the flow network N defined as follows; see also Section 1.4.4 in the preliminaries. The underlying graph of N is the dual graph of the variable-clause graph of S. The demand of the node corresponding to the face f_S is the difference between the rotation in the corresponding large face f of $G(S)$ and 4 (-4 if f is the outer face). Note that the demands sum up to 0. The capacity on an edge connecting f_S and f'_S is equal to the total length of the bendable pipes

incident to the corresponding faces f and f' in $G(S)$, and thus at least K. As shown above, the absolute value of the demand of each node in the flow network is at most $3m + 4$. As the flow network contains at most m nodes (otherwise it would be a tree or disconnected), the sum of the absolute values of the demands is bounded by $3m^2 + 4m$. The capacity of every edge in N is at least $K = 3m^2 + 4m$ by the construction of the variable tree. By Lemma 1.1 the network N has a solution. □

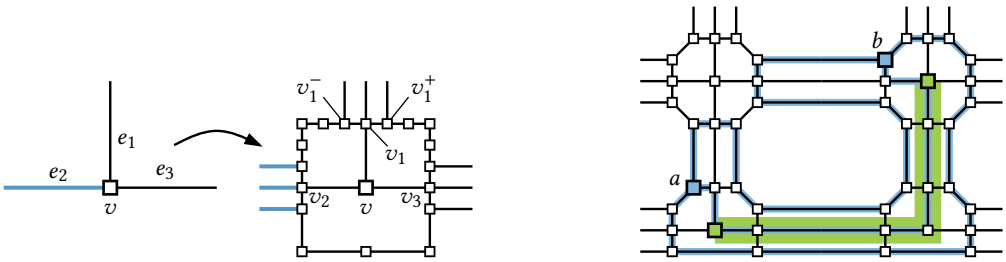
Theorem 4.2. *Orthogonal 01-Embeddability is NP-hard for all combinations of the following variations.*

- *The input has a fixed planar embedding or a fixed planar embedding up to the choice of an outer face.*
- *The angles at vertices incident to* 1*-edges are fixed or variable, while angles at vertices incident to* 0*-edges are variable.*

Proof. In the construction showing Theorem 4.1, we already fixed all angles at vertices incident to 1-edges (the only vertices whose angles are not fixed lie inside interval gadgets). Thus, we already established hardness for the case that all angles at vertices incident to 1-edges are fixed. As mentioned before, fixing angles is not a really a restriction, as we can enforce fixed angles by attaching degree-1 vertices.

It remains to show that the problem remains hard when allowing to choose a different outer face. Clearly, when choosing a different large face as outer face all arguments leading to a satisfying truth assignment remain valid. Moreover, choosing a small face as outer face can never lead to a valid orthogonal 01-representation for the following reason. Each small face is one of the building blocks presented in Section 4.2.1 (see Figure 4.5), or the inner face of a clause gadget (Figure 4.9). For the building blocks it is easy to see that the total rotation in the inner face is at least 0 (by the fixed angles and the restriction of bends on the edges). Thus, none of them can be chosen as the outer face (which would require a rotation of -4). Similarly, the rotations at every vertex in the clause gadget is 1 in its inner face, which sums up to a rotation of 4. The three input edges have rotation at least -1 in the inner face and the interval gadget has rotation at last -2. Thus, the total rotation is at least -1, which makes it impossible to choose it as the outer face. □

By the equivalence of orthogonal representations to flow networks [Tam87], it follows that it is NP-hard to test whether there is a valid flow in a planar flow network with the properties that (i) the capacity on every edge is 1 and (ii) some undirected edges require to have one unit of flow (no matter in which direction). Note that Garg and Tamassia [GT01] show hardness for the less restrictive case that the capacities and the lower bounds for flow on undirected edges is unbounded. They use this to show NP-hardness of Orthogonal 0-Embeddability of 4-planar graph (with variable



(a) (b)

Figure 4.12: (a) Construction for transforming an instance of Orthogonal 01-Embeddability into a subdivision of a 3-connected graph. A vertex v of degree 3 with variable angles (left) and the corresponding gadget for the construction (right). (b) The routing of three disjoint paths from a to b in G'; subdivision vertices are omitted. Vertices u and v are marked green, and the corresponding path in G is bold green. The three blue paths between a and b follow the bold green path. Note that at the beginning at the end of a path some rerouting via vertices not on the path may be necessary, however, the rerouting is such that the paths remain disjoint.

planar embedding); i.e., for the problem of testing whether the given graph admits an orthogonal drawing without bends.

Theorem 4.3. *All variants of Orthogonal 01-Embeddability are NP-hard even if the input graph is a subdivision of a 3-connected graph.*

Proof. We reduce from Orthogonal 01-Embeddability with fixed planar embedding and variable angles. Let $G = (V, E_0 \cup E_1)$ be a connected instance of this problem. We replace each degree-1 vertex v by a cycle C of four 0-edges such that one vertex of C is adjacent to the neighbor of v. It is not hard to see that the resulting graph has an orthogonal 01-embedding if and only if G has one. In the following we assume without loss of generality that G has minimum degree 2.

For each vertex v with incident edges $e_1, \ldots, e_d$ (in clockwise order around v), we make the following construction. First, we subdivide its incident edges e_i with new vertices v_i and connect them to form a cycle (in the clockwise ordering around v), and subdivide the edges of this cycle five times. The vertices before and after v_i in clockwise direction are denoted v_i^- and v_i^+. Afterwards, each edge uv has been subdivided into uu_i, u_iv_j, v_jv. We now add for each such edge the edges $u_i^-v_j^+$ and $u_i^+v_j^-$. The edges u_iv_j, $u_i^-v_j^+$, and $u_i^+v_j^-$ are 1-edges if and only if the original edge uv was a 1-edge. All other edges are 0-edges. Figure 4.12a illustrates the construction for a vertex of degree 3.

We claim that the resulting graph G' (i) admits an orthogonal 01-representation if and only if G does, and (ii) is a subdivision of a 3-connected graph. Once the claim is proved, the statement of the theorem follows since the reduction can be performed in polynomial time.

We start with (i). First assume that G' has an orthogonal 01-representation $\mathcal{O}$ and let uv be an edge of G that is subdivided into u_iv_j. By construction both u_i and v_j have degree 4 and the edges uu_i and v_jv are 0-edges. That is all bends of the path uu_iv_jv lie on the edge u_iv_j. Hence, the representation on the subgraph containing the vertices $\{v, v_1, \dots, v_{\deg(v)} \mid v \in V\}$ has all bends on the edges v_iv_j. We can then undo the subdivisions and obtain an orthogonal 01-representation of G. Conversely, if $\mathcal{O}$ is an orthogonal 01-representation of G, we can first subdivide each edge uv close to vertices u and v to obtain vertices u_i and v_j with uu_i and v_jv having 0 bends. Then we add the edges the edges $u_i^-v_j^+$ and u_i^+, v_j^- parallel to u_iv_j. Finally, we add the remaining edges of the cycle around v, which can be done without bends on the edges since the paths from v_i^+ to v_{i+1}^- (indices taken modulo $\deg(v)$) have sufficiently many degree-2 vertices, which can serve as bends. We have obtained an orthogonal 01-representation of G'.

For (ii), we show that in G' any two vertices a and b of degree 3 or more are connected by three (internally) vertex-disjoint paths. Let u and v be the two vertices of G to whose construction a and b belong. If $u = v$ it is not hard to find three disjoint paths; one path goes through the center vertex v, the remaining paths are routed clockwise and counterclockwise along the cycle around v. It may be necessary to route through a neighboring gadget to get around the attachment vertices of v; see Figure 4.12b.

If $u \neq v$, we pick a shortest path $u = u_1, \dots, u_k = v$ from u to v in G. This path corresponds to a path $u_1a_1b_2u_2, \dots, a_{k-1}b_ku_k$, where a_i and b_i are vertices of the cycle around vertex u_i. We find three disjoint paths from a_1^-, a_1 and a_1^+ to b_k^-, b_k and b_k^+, respectively, simply by taking for each edge u_iu_{i+1} for $1 < i < k-1$ the path from a_i^+ in clockwise direction along the cycle around u_i via b_i^- to a_{i+1}^+, from a_i via the center vertex u_i and b_i to a_{i+1}, and from a_i^+ in counterclockwise direction along the cycle around u_i via b_i^+ to a_{i+1}^-; this is illustrated in the middle vertex of the green path in Figure 4.12b. We also extend these paths by adding edges $a_{k-1}b_k$, $a_{k-1}^+b_k^-$ and $a_{k-1}^-b_k^+$ so that we have disjoint paths from a_1^- to b_k^+, from a_1 to b_k and from a_1^+ to b_k^-. It then remains to find disjoint paths from a to a_1^-, a_1 and a_1^+ and from b_k^-, b_k and b_k^+ to b. This can be done by routing in the gadget around u and v, respectively. Note that it may be necessary to visit the gadget of an adjacent vertex. This does, however, not interfere with the paths constructed so far since we assumed that it is a shortest path, and hence the corresponding neighbors are not part of the constructed paths. This finishes the proof of the claim. □

Corollary 4.1. *Orthogonal 0-Embeddability is NP-hard for 4-planar graphs with a variable planar embedding.*

Proof. We reduce from Orthogonal 01-Embeddability where the input graph is a subdivision of a 3-connected graph. Note that the embedding is unique up to the choice

of the outer face. We now replace each 1-edge by a copy of the interval gadget $G[1,1]$ (see Figure 4.4). Changing the embedding of this gadget decides the bend direction of the 1-edge and vice versa. It is not hard to see that the resulting graph admits a 0-embedding if and only if the original instance admits an orthogonal 01-embedding. Clearly the reduction runs in polynomial time. □

4.2.2 Kandinsky Bend Minimization

In the following, we show how to reduce Orthogonal 01-Embeddability to Kandinsky Bend Minimization. The reduction consists of two basic building blocks. In an orthogonal drawing, every side of a vertex can be occupied by at most one edge. We show how to enforce this requirement also for Kandinsky drawing. Moreover, we construct a subgraph whose Kandinsky drawing behave like the drawings of an edge with exactly one bend.

Corner Blocker. Let B be the graph consisting of a 4-cycle together with an additional *attachment vertex* connected to two non-adjacent vertices of the 4-cycle. The graph B is called *corner blocker*. Figure 4.13a shows a corner blocker with attachment vertex v. Let v be a vertex in a planar graph G. *Blocking a corner of* v denotes the process of attaching a corner blocker to v by identifying the attachment vertex of B with v. Consider a Kandinsky drawing of $G + B$. The corners of the box representing the vertex v are also called the *corners* of v. We say that a corner of v is *blocked* by the corner blocker B if it lies in the inner face of B incident to v. Figure 4.13b shows a vertex v with four corner blockers attached to it such that all four corners of v are blocked. Note that the Kandinsky representation of a Kandinsky drawing already determines which corners are blocked by a corner blocker.

The idea behind the corner blocker is to enforce a blocking of all four corners of a vertex. Recall that we assume a fixed planar embedding of the input graph and thus a fixed order of edges around every vertex. Thus, blocking all four corners is equivalent to enforcing edges to leave a vertex at a specific side, as in Figure 4.13b. The following two lemmas show that the corner blocker defined above is well suited for this purpose, as it admits an optimal drawing blocking only a single corner but blocking no corner causes additional cost.

Lemma 4.7. *Every Kandinsky representation of a corner blocker has at least two bends.*

Proof. Let B be a corner blocker. Denote the degree-2 vertices of B with u, v, and w and the degree-3 vertices with s and t and let B be embedded such that the boundary of the outer face f contains u, v, s, and t; see Figure 4.13a. In every Kandinsky representation, the total rotation around f is -4. We show that this already implies that every Kandinsky representation has at least two bends.

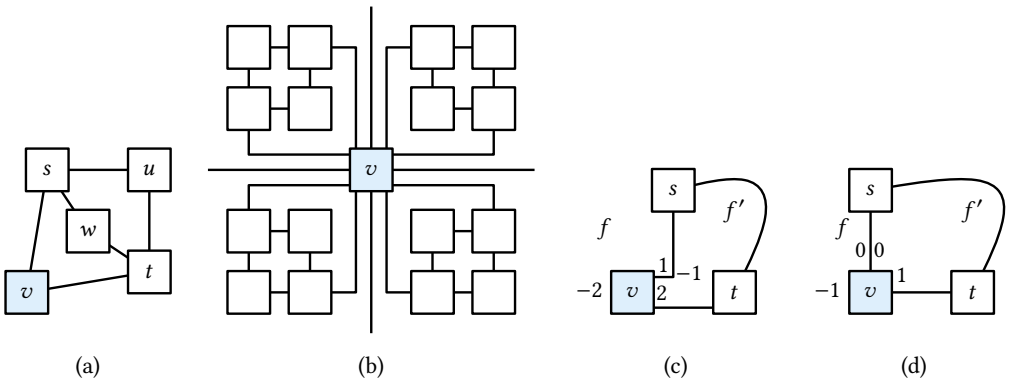


Figure 4.13: (a) A corner blocker with attachment vertex v. (b) A Kandinsky representation of a vertex v with four attached corner blockers (and four outgoing edges). (c–d) Illustration of the proof of Lemma 4.7.

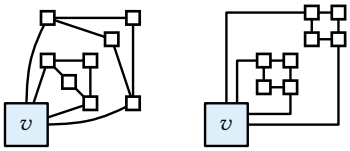
Let f' be the inner face incident to v. If v has rotation 2 in f' for a fixed Kandinsky representation, then one of the two edges vs or vt has rotation -1 at v. We assume without loss of generality that vs has rotation -1 at v, thus we get the following rotation values (see Figure 4.13c): $\text{rot}_{f'}(v) = 2$, $\text{rot}_f(v) = -2$, $\text{rot}_{f'}(vs[v]) = -1$, and $\text{rot}_f(vs[v]) = 1$. As v has degree 2, we obtain another Kandinsky representation by setting $\text{rot}_{f'}(v) = 1$, $\text{rot}_f(v) = -1$, $\text{rot}_{f'}(vs[v]) = 0$, and $\text{rot}_f(vs[v]) = 0$; see Figure 4.13d. As this new Kandinsky representation has fewer bends, we can assume in the following that $\text{rot}_{f'}(v) \neq 2$, which shows that the rotation at v in f is at least -1. Clearly, the same holds for u.

A similar argument shows that the rotations at s and t in f are at least 0. It follows that the total rotation of vertices in the outer face is at least -2. Thus, to get a total rotation of -4, there need to be two bends on edges incident to the outer face, which shows the claim. □

Lemma 4.8. *Every Kandinsky representation of a corner blocker that blocks no corner of its attachment vertex has at least three bends.*

Proof. As shown in the proof of Lemma 4.7, one can reduce the number of bends of a Kandinsky representation of a corner blocker if the rotation at the attachment vertex in the outer face is -2; see Figure 4.13c,d. □

We can make the corner blockers stronger by nesting them. The *nested corner blocker B_d of depth d* is obtained by taking d corner blockers and identifying their attachment vertices. The nested corner blocker B_d is embedded such that v lies on the outer face and the innermost face has distance d to the outer face (in the dual graph); see Figure 4.14 for an example. Clearly, the statements from Lemma 4.7 and Lemma 4.8 extend to nested corner blockers, where all bend numbers have to be multiplied with d.



(a) (b)

Figure 4.14: (a) The nested corner blocker B_2 of depth 2. (b) Kandinsky representation of B_2 with 4 bends.

One-Bend Gadget. Let Γ be the graph consisting of the 2×3-grid with the two columns v_1, v_2, v_3 and u_1, u_2, u_3 (from bottom to top) together with the vertex v connected to v_1, v_2, and v_3 and the vertex u connected to u_2; see Figure 4.15a. We call Γ the *one-bend gadget* with its two *endvertices* u and v. The path $\pi = (u, u_2, v_2, v)$ is called the *bending path* of Γ. In the following we show that the bending path of a one-bend gadget is (more or less) forced to have either rotation 1 or -1 in every Kandinsky representation. As for the corner blocker, we say that the one-bend gadget blocks k corners of the vertex v in a given Kandinsky representation if k corners of v lie in the inner face of Γ.

Lemma 4.9. *Let $\mathcal{K}$ be a bend-minimal Kandinsky representation of the one-bend gadget Γ blocking no corner of its degree-3 end vertex. Then $\mathcal{K}$ has three bends and the rotation of the bending path in Γ is either 1 or -1.*

Proof. Note that the Kandinsky representation of Γ in Figure 4.15a does not block a corner of v and has three bends. It remains to show that this drawing is optimal and that the rotation of the bending path π is always 1 or -1.

We consider all Kandinsky representations of Γ with at most three bends blocking no corner of v and show that each of these representations has three bends and rotation 1 or -1 on π. We start with two simple facts. First, blocking no corner of v requires that at least two of the edges vv_1, vv_2, and vv_3 to have a bend, as they all leave v at the same side. Second, the edges in each of the triangles vv_1v_2 and vv_2v_3 require at least two bends, as they have rotation 2 at v. We use these facts several times in the following case distinction on the number of bends of vv_2.

Assume that $\boldsymbol{vv_2}$ **has three bends**. As at least two of the edges vv_1, vv_2, and vv_3 have bends, we get at least four bends in total (but we consider only drawings with at most three bends). Assume that $\boldsymbol{vv_2}$ **has zero bends**. Then the two bends of the triangle vv_1v_2 must be on the edges vv_1 and v_1v_2 and the bends of the triangle vv_2v_3 must be on the edges vv_3 and v_2v_3. Thus, there are at least four bends, which again contradicts the restriction to at most three bends.

If $\boldsymbol{vv_2}$ **has two bends**, one of the two edges vv_1 or vv_3 has one bend, the other has no bend (as we have more than three bends otherwise). Assume that vv_3 has one bends, the other case is symmetric. As vv_1 has no bend, the direction of the bend of vv_3 and of the first bend of vv_2 is fixed. The remaining choice is the second bend of vv_2; see Figure 4.15b and c for an illustration of the two possible Kandinsky representations.

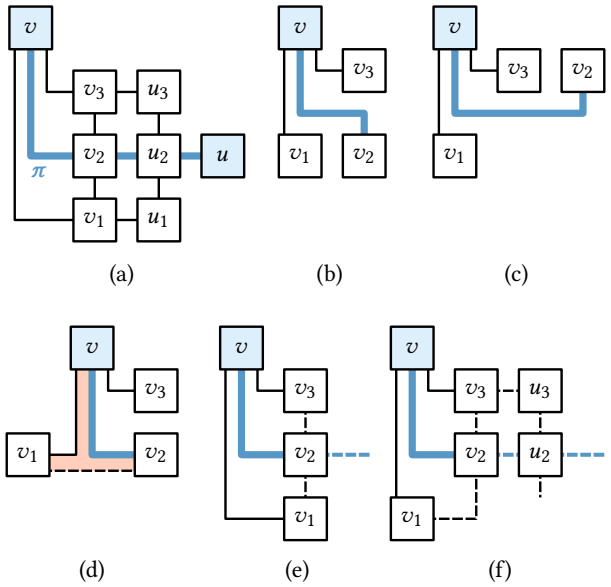


Figure 4.15: (a) The one bend gadget Γ with three bends without blocked corners at v with $|\operatorname{rot}(\pi)| = 1$. (b–c) The two possible representations of the edges incident to v, when vv_2 (blue) has two bends. Both cannot be extended to Γ without additional bends. (d–f) The possible ways to draw vv_1, when vv_2 (blue) has one bend together with the implied drawing on some other edges (dashed). Either we get no Kandinsky representation of Γ (d), or the path π has absolute rotation 1 (e–f).

Since we already used three bends, the 2×3 grid consisting of the nodes $v_1 \dots v_3$ and $u_1 \dots u_3$ must be drawn without any bends. However, the Kandinsky representation without bends of the 2×3 grid is unique (see Figure 4.15a) and can obviously not be merged with one of the Kandinsky representations of the three edges incident to v shown in Figure 4.15b and c.

Assume that $\boldsymbol{vv_2}$ **has one bend**. Assume without loss of generality that the bend on vv_2 is a left bend, when traversing it from v to v_2 (i.e., vv_2 has rotation 1 in the triangle vv_2v_3). This implies that the edge vv_3 has at least one bend as in Figure 4.15d–f. If vv_1 has a bend but in the other direction then all remaining edges have to be straight, which is not possible for v_1v_2 without creating an empty triangle (Figure 4.15d). Thus, v_1v_2 has either a bend in the same direction as vv_1 or no bend. Consider the former case first; see Figure 4.15e. We split the bending path π into two parts, the edge vv_2 and the path from v_2 to u. Clearly, the absolute rotation of vv_2 is 1. As we already used three bends, the Kandinsky representation of $\Gamma - v$ is unique. Thus, the path from v_2 to u must have rotation 0. To show $|\operatorname{rot}(\pi)| = 1$, it remains to show that the rotation of π at v_2 is 0, which is the case if there is no 0° angle at v_2. This angle would have to be adjacent to the edge vv_2 as it is the only one having a bend. However, vv_2 has only one bend and, since $\operatorname{rot}_f(vv_2[v]) = 1$, it follows that $\operatorname{rot}_f(vv_2[v_2]) = 0$, where f is the face bounded by vv_2v_3. But then there can be no 0° bend at v_2.

It remains to deal with the case that vv_1 has no bend; see Figure 4.15f. As the triangle vv_1v_2 needs two bends, the edge v_1v_2 must be drawn with a bend. All remaining edges must have zero bends, as three bends are already used. As before, this shows that the subpath of π from v_2 to u has rotation 0 and for $|\operatorname{rot}(\pi)| = 1$ it remains to

show that the rotation of π at v_2 is 0. To this end, consider the triangle vv_2v_3 and the quadrangle $v_2u_2u_3v_3$. In the quadrangle, all edges are straight lines, which ensures that the rotation at v_2 is 1. In the triangle, the rotation at v_2 must also be 1 (otherwise the rotation at v_3 would need to be 2, but there is no edge that can assign its bend to this 0° angle). Thus, the rotation of v_2 in the path π is 0, which shows $|\operatorname{rot}(\pi)| = 1$.

It follows that the path π has rotation 1 or -1 in every Kandinsky representation of Γ that has only three bends and blocks no corner of v. Moreover, we showed that all such Kandinsky representations require three bends. □

Putting Things Together

Let $G = (V, E = E_0 \cup E_1)$ (together with a planar embedding) be an instance of Orthogonal 01-Embeddability. We assume that the angles at vertices that are incident to a 1-edge are fixed. We construct an embedded graph G' that then serves as instance of Kandinsky Bend Minimization. To construct G', we start with G. Let v be a vertex incident to the face f. If the angle of v in f is fixed to α, we attach $\alpha/90°$ nested corner blockers of depth 4 to v embedded next to each other into the face f; see Figure 4.16a. Otherwise, if the angle is not fixed, we attach a single corner blocker of depth 4 at v in f. By suitably increasing the depth of some corner blockers we ensure that each vertex is incident to exactly 16 corner blockers; see Figure 4.16b. This is not strictly necessary but simplifies some of our computations. Finally, we replace every edge $uv \in E_1$ (i.e., every edge that requires one bend) by a copy of the one-bend gadget Γ, identifying u and v with the endvertices of Γ. Note that, by assumption, both u and v have four corner blockers of depth 4. To obtain the following theorem, we show that the resulting graph G' admits a Kandinsky representation with at most $32|V| + 3|E_1|$ bends if and only if G admits an orthogonal 01-embedding (note that deciding whether a planar embedded graph admits a Kandinsky representation with at most k bends is clearly in NP).

Theorem 4.4. *Kandinsky Bend Minimization is NP-complete.*

Proof. Let G be an instance of Orthogonal 01-Embeddability (with fixed angles at vertices incident to 1-edges) and let G' be the corresponding instance of Kandinsky Bend Minimization. Assume we have an orthogonal 01-representation $\mathcal{O}$ of G. We show how to construct an Kandinsky representation $\mathcal{K}$ of G' with $32|V| + 3|E_1|$ bends. We interpret $\mathcal{O}$ as a Kandinsky representation. We first add the nested corner blockers to the representation. Let v be a vertex with incident face f. By construction, v has at least as many corners in f as there are nested corner blockers incident to v embedded in the face f. Thus, these corner blockers can be added with 2 bends for each corner blocker (see the drawing in Figure 4.13b). This yields $32|V|$ bends in total.

Moreover, the drawings of the 1-edges can be replaced by drawings of one-bend gadgets with three bends (the drawing in Figure 4.15a or the symmetric drawing

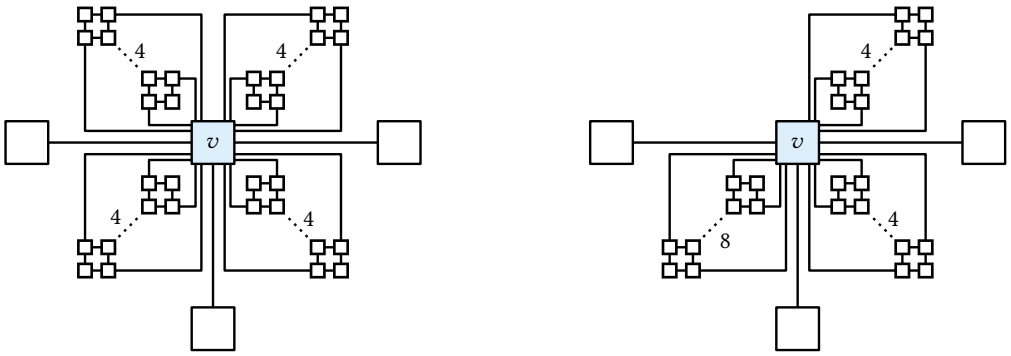


Figure 4.16: (a–b) A degree-3 vertex with and without fixed angles and attached corner blockers, respectively.

where the bending path π is bent to the other direction). This yields $3|E_1|$ bends for all 1-edges. Hence, we get a Kandinsky representation $\mathcal{K}$ of G' with $32|V| + 3|E_1|$ bends in total.

For the opposite direction, we show that a Kandinsky representation $\mathcal{K}$ of G' with at most $32|V| + 3|E_1|$ bends implies the existence of a orthogonal 01-embedding $\mathcal{O}$ of G. We show that the following three facts hold for $\mathcal{K}$.

1. Every nested corner blocker blocks a corner.
2. Every one-bend gadget has three bends and blocks no corner of its degree-3 endvertex in $\mathcal{K}$.
3. All remaining edges (the edges in E_0) have 0 bends.

We use a charging argument assigning the costs for bends either to corner blockers, to one-bend gadgets or to the edges in E_0, such that the total cost is at most the total number of bends. By Lemma 4.7, every corner blocker of depth d requires at least $2d$ bends. Moreover, if such a nested corner blocker does not block a corner, it has at least $3d$ bends (Lemma 4.8). For corner blockers that block a corner, we charge cost $2d$ (which is equal to the number of bends). For corner blockers blocking no corner, we charge cost $2d + 1$ (which is $d - 1 \geq 3$ less than the number of bends; note that all corner blockers have depth at least 4). A one-bend gadget with more than three bends is charged cost 4. A one-bend gadget that does not block a corner of its degree-3 endvertex has at least three bends by Lemma 4.9 and we charge cost 3 for it. If a one-bend gadget blocks a corner of its degree-3 endvertex, then at least one of the adjacent nested corner blockers does not block a corner. As we charged cost $2d + 1$ for this corner blocker although it has at least $3d$ bends, we can again

charge cost $3d - (2d + 1) = d - 1 \geq 3$ for the one-bend gadget. For the remaining edges in E_0 we simply charge cost equal to the number of bends.

Hence, every nested corner blocker of depth d is charged at least cost $2d$ and every one-bend gadget is charged at least cost 3. Recall that there are $16|V|$ corner blockers and $|E_1|$ one bend gadgets. To get a total cost of at most $32|V| + 3|E_1|$, every corner blocker of depth d must be charged exactly cost $2d$, which implies that it blocks a corner and thus shows the first fact. Since the endvertices of one-bend gadgets are incident to four corner blockers, each of which indeed blocks a corner, this also implies that no one-bend gadget can block a corner of its degree-3 endvertex. Thus, by Lemma 4.9, every one-bend gadget has at least three bends. Moreover, every one-bend gadget has no more than three bends as it would otherwise be charged cost 4, which shows the second fact. The third fact follows as the cost charged to edges in E_0 must be 0.

By the second fact and Lemma 4.9 the bending path of every one-bend gadget has absolute rotation 1 in $\mathcal{K}$. Thus, we can replace each one-bend gadget by an edge with exactly one bend. Removing the corner blockers yields a representation of G in which no two edges leave a common incident vertex on the same side, as every nested corner blocker blocks a corner (first fact). Moreover, the edges in E_0 have zero bends. Hence, the resulting representation of G is an orthogonal representation (and not only a Kandinsky representation) and the edges in E_1 and E_0 have one and zero bends, respectively. □

Theorem 4.5. *Kandinsky Bend Minimization is NP-complete, even if we allow empty faces or require every edge to have at most one bend (or both).*

Proof. That the problem remains NP-hard when we require each edge to have at most one bend is obvious, as all Kandinsky representations involved in the construction above have at most one bend per edge. In fact, this requirement would even make some arguments simpler. The only place where we argued with empty faces is in the proof of Lemma 4.9 to exclude the situation shown in Figure 4.15d. It is not hard to see that this situation can also be excluded when allowing empty faces, as even in this case, it is not possible to complete the drawing without additional bends. □

4.3 A Subexponential Algorithm

In this section, we give an algorithm for computing optimal Kandinsky representations of planar graphs with fixed planar embedding in subexponential running time. To this end, we use dynamic programming on sphere cut decompositions, which are special types of branch decompositions [Dor+10].

The basic idea is as follows. Consider two graphs G_1 and G_2 with disjoint edge sets that share a set of *attachment vertices*. We assume that the union G of G_1 and G_2 is planar and has a fixed planar embedding. We say that G_1 and G_2 are *glueable* if both

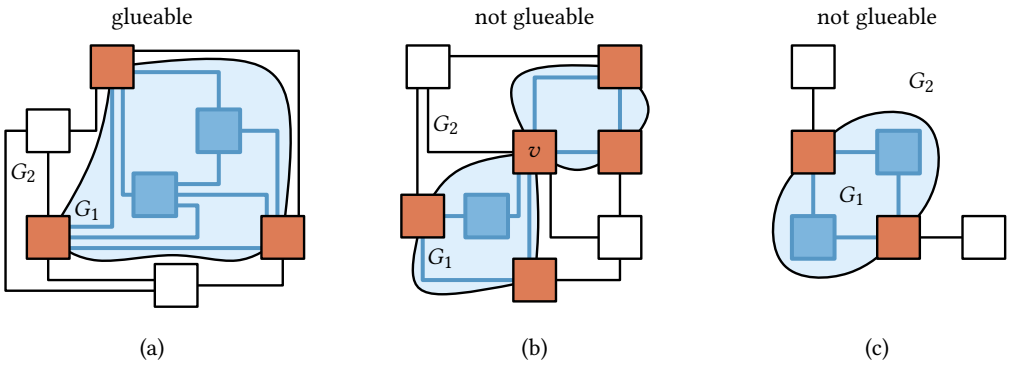


Figure 4.17: (a) The decomposition of a graph into two glueable subgraphs G_1 and G_2. The attachment vertices are red. (b) This decomposition is not glueable, as a closed curve separating G_1 from G_2 cannot be simple (v must be visited twice). (c) The decomposition is not glueable since G_2 is disconnected.

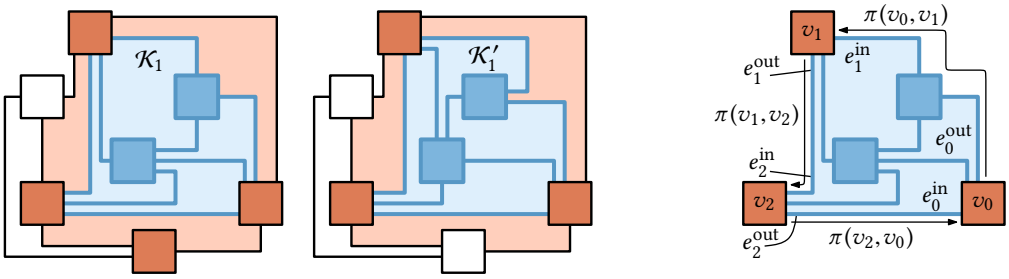
graphs are connected and there is a simple closed curve in the embedding of G that separates G_1 from G_2 (note that this curve must contain the attachment vertices); see Figure 4.17. We also say that G_1 (G_2) is a *glueable subgraph* of G.

Now assume that we know two Kandinsky representations $\mathcal{K}_1$ and $\mathcal{K}_2$ of G_1 and G_2. Depending on $\mathcal{K}_1$ and $\mathcal{K}_2$ one might be able to merge them into a Kandinsky representation of the whole graph G. We can generate every Kandinsky representation of G in this way, by merging every representation of G_1 with every representation of G_2. Clearly, considering all pairs of representations of G_1 and G_2 is not efficient. Thus, we group Kandinsky representations of G_1 that behave the same with respect to merging them with representations of G_2 into equivalence classes. If we know an optimal Kandinsky representation for each equivalence class of G_1 and G_2, it is sufficient to merge those optimal representatives of equivalence classes to obtain an optimal representation of G. If G is hierarchically decomposed, one can start with optimal Kandinsky representations of the edges and merge them step by step to obtain G.

In the following we first characterize which Kandinsky representations of a glueable subgraph are equivalent in the sense that they can be merged with the same Kandinsky representation of the remaining graph (Section 4.3.1). Afterwards, we estimate in how many different ways the Kandinsky representations of subgraphs can be merged into one (Section 4.3.2). Finally, we conclude with the algorithm and some interesting special cases (Section 4.3.3).

4.3.1 Interfaces of Kandinsky Representations

Let $\mathcal{K}$ be a Kandinsky representation of G and let $\mathcal{K}_1$ be the representation induced on G_1. Let $\mathcal{K}_1'$ be another Kandinsky representation of G_1. By *replacing* $\mathcal{K}_1$ *with* $\mathcal{K}_1'$ *in*



(a) (b)

Figure 4.18: (a) The graph G with glueable subgraph G_1 (shaded blue). The attachment vertices and the faces shared by G_1 and G_2 are red. The two Kandinsky representations $\mathcal{K}_1$ (left) and $\mathcal{K}_1'$ (right) of G_1 are interchangeable: In any representation of G inducing $\mathcal{K}_1$ on G_1, one can replace $\mathcal{K}_1$ by $\mathcal{K}_1'$ and vice versa. (b) Illustration of the notation used to define the interface paths, the attachment rotations and the 0° flags.

$\mathcal{K}$ we mean the following. Every rotation value in $\mathcal{K}$ involving only edges belonging to G_1 are set to the value specified in $\mathcal{K}_1'$ while all other values remain as they are. In other words the following rotations in $\mathcal{K}$ are changed to their value in $\mathcal{K}_1'$: $\text{rot}(e)$ if the edge e belongs to G_1; $\text{rot}(uv[u])$ and $\text{rot}(uv[v])$ (the rotation of uv at the vertices u and v) if uv belongs to G_1; and $\text{rot}(e_1, e_2)$ (for two edges e_1 and e_2 incident to a common vertex) if both edges e_1 and e_2 belong to G_1. Note that the resulting set of rotation values is not necessarily a Kandinsky representation, as some properties of Kandinsky representations might be violated.

We say that the two Kandinsky representations $\mathcal{K}_1$ and $\mathcal{K}_1'$ of G_1 have *the same interface* if replacing $\mathcal{K}_1$ with $\mathcal{K}_1'$ (and vice versa) in any Kandinsky representation of G yields a Kandinsky representation of G. We will see later (Lemma 4.10) that it does not depend on the remaining graph $G \setminus G_1$, whether two representations of G_1 have the same interface. The two Kandinsky representations in Figure 4.18a have the same interface. Clearly, having the same interface is an equivalence relation. We call the equivalence classes of this relation the *interface classes*.

Now consider again two glueable subgraphs G_1 and G_2 of a plane graph G. Since G_1 and G_2 are glueable, we know that G_2 lies in a single face f of G_1. Let C_f be the facial cycle of f and assume for now, that C_f is simple (i.e., G_1 contains no cutvertex incident to f). Let $v_0, \dots, v_\ell$ be the attachment vertices appearing in that order in C_f (clockwise for inner, counter-clockwise for outer faces). This decomposes C_f into the paths $\pi_f(v_0, v_1), \pi_f(v_1, v_2), \dots, \pi_f(v_\ell, v_0)$, which we call *interface paths*. As the face we consider is unique, we often omit the subscript and simply write $\pi(v_i, v_{i+1})$. Moreover, the values for i, $i-1$ and $i+1$ are always meant modulo $\ell+1$. For an attachment vertex v_i, denote the last edge of the path $\pi(v_{i-1}, v_i)$ by e_i^{in} and the first edge of the path $\pi(v_i, v_{i+1})$ by e_i^{out}; see Figure 4.18b.

Let $\mathcal{K}_1$ and $\mathcal{K}_1'$ be two Kandinsky representations of G_1. We say that $\mathcal{K}_1$ and $\mathcal{K}_1'$ have *compatible interface paths* if $\pi(v_i, v_{i+1})$ has the same rotation in $\mathcal{K}_1$ and $\mathcal{K}_1'$ (i.e., $\text{rot}_{\mathcal{K}_1}(\pi(v_i, v_{i+1})) = \text{rot}_{\mathcal{K}_1'}(\pi(v_i, v_{i+1}))$) for every $i = 1, \ldots, k$. Moreover, $\mathcal{K}_1$ and $\mathcal{K}_1'$ have *the same attachment rotations* if for every attachment vertex v_i, the rotation $\text{rot}(e_i^{\text{in}}, e_i^{\text{out}})$ is the same in $\mathcal{K}_1$ and $\mathcal{K}_1'$. In Figure 4.18b, the interface paths $\pi(v_0, v_1)$, $\pi(v_1, v_2)$, and $\pi(v_2, v_0)$ have rotations -1, 1, and 0, respectively, and the attachment rotations at the vertices v_0, v_1, and v_2 are -1, -1, and -2, respectively.

When considering orthogonal representations (with maximum degree 4) and not Kandinsky representations, having compatible interface paths and the same attachment rotations is sufficient for two representations to have the same interface. In case of Kandinsky representations, we have to care about $0°$ angles at the attachment vertices. Thus, for an attachment vertex v_i, the rotations at the end v_i of the edges e_i^{in} and e_i^{out} ($\text{rot}(e_i^{\text{in}}[v_i])$ and $\text{rot}(e_i^{\text{out}}[v_i])$), which can take the values -1, 0, or 1 are of importance. The actual value of these rotations is not important, we only care about whether they are -1 or something else. We call these information the $0°$ *flags*, which has the value `true` for a rotation of -1 and `false` otherwise. We say that $\mathcal{K}_1$ and $\mathcal{K}_1'$ have the *same* $0°$ *flags* if all their $0°$ flags have the same values. Possible values for the $0°$ flags in Figure 4.18b are `true` for $e_0^{\text{out}}[v_0]$ and for $e_1^{\text{in}}[v_1]$ and `false` for all other flags.

In case the facial cycle C_f is not simple, it might contain an attachment vertex v_i several times. However, since G_1 and G_2 are glueable, the simple closed curve separating G_1 from G_2 gives an order of the attachment vertices. We simply take this order to define the interface paths. All remaining definitions work as before.

Lemma 4.10. *Two Kandinsky representations have the same interface if and only if they have compatible interface paths, the same attachment rotations, and the same* $0°$ *flags.*

Proof. We first show the only-if part. Let G be a plane graph with Kandinsky representation $\mathcal{K}$ with restriction $\mathcal{K}_1$ to the glueable subgraph G_1. Let $\mathcal{K}_1'$ be another Kandinsky representation of G_1. Assume there is an interface path π that has a different rotation in $\mathcal{K}_1$ than in $\mathcal{K}_1'$. Let f be the face incident to π shared by G_1 and the remaining graph G_2 (one of the red faces in Figure 4.18a). By replacing $\mathcal{K}_1$ with $\mathcal{K}_1'$ the rotation of π in f changes, but all other rotations in f stay the same. Thus, the total rotation around f cannot be 4 (-4 if f is the outer face), which shows that the resulting set of rotations is not a Kandinsky representation of G (contradiction to Property (1)). Hence, $\mathcal{K}_1$ and $\mathcal{K}_1'$ do not have the same interface. A similar argument shows that having the same attachment rotations is necessary, since otherwise the total rotation around a vertex would change by replacing $\mathcal{K}_1$ with $\mathcal{K}_1'$, which contradicts Property (3).

Finally, assume that $\mathcal{K}_1$ and $\mathcal{K}_1'$ have different $0°$ flags. Thus, there exists an attachment vertex v with incident edge e_1 (belonging to an interface path) such that $\text{rot}(e_1[v])$ is (without loss of generality) -1 in $\mathcal{K}_1$ (value `true`) and 0 or 1 in $\mathcal{K}_1'$ (value `false`). As v is an attachment vertex, the remaining graph G_2 contains an edge incident to v. Let e_2 be the edge of G_2 incident to v that shares a face f with e_1. Then one

might choose the Kandinsky representation $\mathcal{K}$ of G such that the rotation $\text{rot}_f(e_1, e_2)$ at v in f is 2 (angle of 0°) while $\text{rot}_f(e_2[v])$ is 0 or 1. Then $\text{rot}(e_1[v])$ must be -1 by Property (5), which is true for $\mathcal{K}_1$ but not for $\mathcal{K}_1'$. Hence, replacing $\mathcal{K}_1$ with $\mathcal{K}_1'$ does not yield a Kandinsky representation of G, which shows that having the same 0° flags is also necessary for having the same interface.

For the other direction, let G_1 and G_2 be glueable graphs with union G and let $\mathcal{K}$ be a Kandinsky representation of G with restrictions $\mathcal{K}_1$ and $\mathcal{K}_2$ to G_1 and G_2, respectively. Let $\mathcal{K}_1'$ be a Kandinsky representation of G_1 with compatible interface paths, the same attachment rotations, and the same 0° flags. We show that replacing $\mathcal{K}_1$ with $\mathcal{K}_1'$ in $\mathcal{K}$ yields a Kandinsky representation of G by showing that the resulting rotation values satisfy properties (1)–(5) from Section 1.4.5.

Property (2) is trivially satisfied, as all rotations concerning a single edge come either from $\mathcal{K}_1'$ or from $\mathcal{K}_2$, which are both Kandinsky representations and thus satisfy this property. Property (4) is also satisfied, as the rotation at a vertex either stays as it is in $\mathcal{K}$ or it is changed to its value in $\mathcal{K}_1'$ and thus lies in the interval $[-2, 2]$.

For Property (1), consider a face f of G. If all edges in the boundary of f belong to only one of the graphs G_1 and G_2, then the total rotation in f is equal to its total rotation in $\mathcal{K}_1'$ or $\mathcal{K}_2$, respectively. As $\mathcal{K}_1'$ and $\mathcal{K}_2$ are Kandinsky representations, they satisfy Property (1). If the boundary of f contains edges from both graphs G_1 and G_2 (one of the red faces in Figure 4.18a), it is composed of two interface paths π_1 and π_2 belonging to G_1 and G_2, respectively, that share their endvertices u and v. By replacing $\mathcal{K}_1$ with $\mathcal{K}_1'$, the representation of π_2 does not change. Moreover, the rotations at u and v in f remain unchanged. The representation of π_1 might of course change, however, the rotation remains the same as $\mathcal{K}_1$ and $\mathcal{K}_1'$ have compatible interface paths.

A similar argument shows that Property (5) is satisfied. Let v be a vertex with rotation 2 (corresponding to an angle of 0°) in a face f, i.e., $\text{rot}_f(uv, vw) = 2$. We only need to consider the case where (without loss of generality) uv belongs to G_1 and vw belongs to G_2, as all other cases are trivial. Then $\text{rot}_f(uv[v]) = -1$ or $\text{rot}_f(vw[v]) = -1$ holds in $\mathcal{K}$. In the latter case, $\text{rot}_f(vw[v])$ does not change by replacing $\mathcal{K}_1$ with $\mathcal{K}_1'$ as vw belongs to G_2. In the former case, $\text{rot}_f(uv[v]) = -1$ implies that the corresponding 0° flag in $\mathcal{K}_1$ is `true`. As $\mathcal{K}_1$ and $\mathcal{K}_1'$ have the same 0° flags, this flag is also `true` in $\mathcal{K}_1'$, which implies that $\text{rot}_f(uv[v]) = -1$ is still true in $\mathcal{K}_1'$ and thus in $\mathcal{K}'$.

Finally, to show Property (3), consider a vertex v. If v is not an attachment vertex, all rotations at v come either from $\mathcal{K}_1'$ or from $\mathcal{K}_2$ and thus satisfy Property (3). Let v be an attachment vertex and let f_1 be the face of G_1 that completely contains G_2. The only rotations at v that might change by replacing $\mathcal{K}_1$ with $\mathcal{K}_1'$ are the rotations in faces not shared with G_2. These are exactly the faces of G_1 incident to v except for f_1. As $\mathcal{K}_1$ and $\mathcal{K}_1'$ have the same rotations at attachment vertices, the rotation $\text{rot}_{f_1}(v)$ is the same in $\mathcal{K}_1$ and $\mathcal{K}_1'$. Thus, by Property (3) the sum of all other rotations around v in G_1 must also be the same in both representations $\mathcal{K}_1$ and $\mathcal{K}_1'$. Hence, the total

sum of rotations at v does not change by replacing $\mathcal{K}_1$ with $\mathcal{K}_1'$, which concludes the proof. □

It follows that each interface class is uniquely described by the rotations of the interface paths, by the rotations at the attachment vertices, and by the values of the $0°$ flags. We simply call this set of information the *interface* of G_1 (G_2) in G. Note that this redefines what it means for two Kandinsky representations to have the same interface. However, the definitions are consistent due to Lemma 4.10 and we will use them interchangeably.

4.3.2 Merging two Kandinsky Representations

So far, we considered the case that there is a Kandinsky representation $\mathcal{K}$ of G that can be altered by replacing the Kandinsky representation of the subgraph G_1. Now we change the point of view and assume that we have Kandinsky representations $\mathcal{K}_1$ and $\mathcal{K}_2$ of G_1 and G_2, respectively, that we want to combine to get a Kandinsky representation of G. We say that $\mathcal{K}_1$ and $\mathcal{K}_2$ can be *merged* if there exists a Kandinsky representation $\mathcal{K}$ of G whose restrictions to G_1 and G_2 are $\mathcal{K}_1$ and $\mathcal{K}_2$, respectively. Note that the only rotations in $\mathcal{K}$ that occur neither in $\mathcal{K}_1$ nor in $\mathcal{K}_2$ are rotations at attachment vertices between an edge of G_1 and an edge of G_2. We call these rotations the *shared rotations*; see Figure 4.19a. Thus, merging $\mathcal{K}_1$ and $\mathcal{K}_2$ is the process of choosing values for the shared rotations, such that the resulting set of rotations is a Kandinsky representation of G.

In the following, we consider the case where G itself is a glueable subgraph of a larger graph H. We call this the *merging step* $G = G_1 \sqcup G_2$. Note that G_1 and G_2 are not only glueable subgraphs of G but also of H. Note further that the interface of G_1 (G_2) in G can be deduced from the interface of G_1 (G_2) in H. When dealing with a merging step, we always consider the interfaces of G_1 and G_2 in H (which contain more information than their interfaces in G). The *width* of a merging step is the maximum number of attachment vertices of G_1, G_2, and G in H; see Figure 4.19b for an example.

If the Kandinsky representations $\mathcal{K}_1$ and $\mathcal{K}_2$ can be merged, then every Kandinsky representation $\mathcal{K}_1'$ with the same interface as $\mathcal{K}_1$ can be merged in the same way (i.e., with the same shared rotations) with $\mathcal{K}_2$ as one can first merge $\mathcal{K}_1$ with $\mathcal{K}_2$ and then replace $\mathcal{K}_1$ by $\mathcal{K}_1'$. Moreover, the resulting Kandinsky representations $\mathcal{K}$ and $\mathcal{K}'$ of G have the same interface for the following reason. In every Kandinsky representation of H the representation $\mathcal{K}$ can be replaced by $\mathcal{K}'$ as this is equivalent to replacing $\mathcal{K}_1$ by $\mathcal{K}_1'$ (which can be done as $\mathcal{K}_1$ and $\mathcal{K}_1'$ have the same interface). Thus, the only choices that matter when merging two Kandinsky representations are to choose shared rotations and interfaces for G_1 and G_2. Thus, the term of merging Kandinsky representations extends to *merging interfaces*. We call a choice of shared rotations and

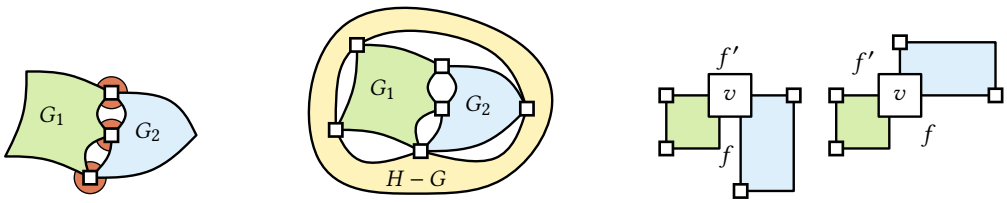


Figure 4.19: (a) Merging G_1 and G_2. The shared rotations are marked red. (b) Illustration of a merging step. The width of this merging step is 5 (G_1 has 5 attachment vertices). (c) Two ways to choose the shared rotations.

interfaces for G_1 and G_2 *compatible*, if these interfaces can be merged using the chosen rotations.

The following lemma bounds the number of compatible combinations. It is parameterized with the width k of the merging step and the *maximum rotation* ρ. The maximum rotation of a graph H is ρ if H admits an optimal Kandinsky representation such that the absolute rotations of the interface paths in every glueable subgraph of H are at most ρ. With the maximum rotation of a merging step, we mean the maximum rotation of the whole graph H. We give bounds for ρ in Lemma 4.13.

Lemma 4.11. *In a merging step $G = G_1 \sqcup G_2$ of width k with maximum rotation ρ, there are at most $(2\rho + 1)^{\lfloor 1.5k \rfloor - 1} \cdot 330^k$ compatible choices for the shared rotations and the interfaces of G_1 and G_2.*

Proof. Let k_{12} be the number of attachment vertices shared by G_1 and G_2 and let k_1 and k_2 be the number of exclusive attachment vertices of G_1 and G_2, respectively. In the example in Figure 4.19b, $k = 5$, $k_{12} = 3$, $k_1 = 2$, and $k_2 = 1$. As G_1 and G_2 both have at most k attachment vertices, we have $k_1 + k_{12} \leq k$ and $k_2 + k_{12} \leq k$. Moreover, every exclusive attachment vertex is an attachment vertex of G, thus $k_1 + k_2 \leq k$ holds. By summing these three inequalities we directly get $k_1 + k_2 + k_{12} \leq \lfloor 1.5k \rfloor$. We start with a rough estimation of the possible combinations and then show how to reduce the number by ruling out choices that are not compatible and thus will never lead to a Kandinsky representation.

The graph G_1 has $k_1 + k_{12} \leq k$ attachment vertices and thus also $k_1 + k_{12} \leq k$ interface paths. The absolute rotation of each interface path is at most ρ, thus there are at most $2\rho + 1$ possible values for those rotations. This leads to at most $(2\rho + 1)^k$ combinations. For every attachment vertex there is the attachment rotation that can be any of the five integers in $[-2, 2]$. Moreover, there are two binary $0°$ flags for each attachment vertex which gives $5 \cdot 2 \cdot 2 = 20$ possible configurations for each attachment vertex. Thus, there are up to 20^k combinations for the $k_1 + k_{12} \leq k$ attachment vertices in G_1. We get the same bounds for G_2. Hence, there are at most $(2\rho + 1)^{2k} \cdot 400^k$

combinations for choosing an interface for G_1 and G_2. For every shared attachment vertex, there are two shared rotations we need to set, which gives 25 combinations as these rotations can take values in $[-2, 2]$. For the k_{12} shared rotations, this gives $25^{k_{12}} \leq 25^k$ combinations, which makes $(2\rho + 1)^{2k} \cdot 10000^k$ combinations in total.

We start with the exponent in the factor $(2\rho + 1)^{2k}$. The exponent $2k$ came from the fact that we chose rotations of $k_1 + k_{12} + k_2 + k_{12}$ interface paths. Assume we have fixed the interface of G_1 except for the rotation of a single interface path. As the total rotation around the face bounded by the interface paths is 4 (-4 for the outer face) in every Kandinsky representation, there is no choice left for the rotation of this path. Thus, we only have to choose the rotation of $k_1 + k_{12} - 1$ interface paths in G_1. The same holds for G_2, which gives $k_1 + k_{12} + k_2 + k_{12} - 2$ interface paths in total. As there are k_{12} shared attachment vertices, the graph G has $k_{12} - 1$ faces that are bounded by one interface path of G_1 and one interface path of G_2. Assume the rotation of the interface paths of G_1 is fixed and the shared rotations are fixed. Then the rotations of these $k_{12} - 1$ interface paths of G_2 are also fixed as the rotation around these faces must sum to 4 (-4). Thus, there are $k_{12} - 1$ additional interface paths whose rotation is automatically fixed. Hence, we get the exponent down to $k_1 + k_2 + k_{12} - 1$ which is at most $\lfloor 1.5k \rfloor - 1$.

To reduce the basis of the 10000^k factor, first note that some configurations of choosing attachment rotations and $0°$ flags are not possible. Let f be the face of G_1 containing all attachment vertices and let v be an attachment vertex. Let e^{in} and e^{out} be the two edges incident to v and f, i.e., $\text{rot}_f(e^{\text{in}}, e^{\text{out}})$ is the attachment rotation at v. Assume $\text{rot}_f(e^{\text{in}}, e^{\text{out}}) = 2$, i.e., there is an angle of $0°$ at v. Due to Property (5), e^{in} or e^{out} must have a rotation of -1 at the vertex v in f ($\text{rot}_f(e^{\text{in}}[v]) = -1$ or $\text{rot}_f(e^{\text{out}}[v]) = -1$). Thus if the attachment rotation at v is 2, at least one of the two $0°$ flags at v must be `true`. A similar argument shows that an attachment rotation of -2 at v implies that at least one of the $0°$ flags at v is `false`. Thus, there are only 18 (instead of 20) possibilities for choosing the attachment rotation and the $0°$ flags at an attachment vertex. Thus, for the exclusive attachment vertices in G_1 and G_2 we get $18^{k_1+k_2}$ combinations. Moreover, we have $18^{2k_{12}}$ combinations for the shared attachment vertices in G_1 and G_2 and $25^{k_{12}}$ combinations for the shared rotations.

We show that not all these $18^{2k_{12}} \cdot 25^{k_{12}}$ need to be considered. Let v be a shared attachment vertex and let rot_1 and rot_2 be the attachment rotations for v in G_1 and G_2, respectively. Let further f and f' be the two faces incident to v shared by G_1 and G_2 and let rot_f and $\text{rot}_{f'}$ be the corresponding shared rotations at v in f and f'. Finally, let x_f ($x_{f'}$) be a variable with the value 1 if the $0°$ flags do not allow a $0°$ angle in f (f') and the value 0 if they allow a $0°$ angle, which is the case if and only if at least one of the corresponding flags is `true`. It is not hard to see, that fixing the attachment rotations rot_1 and rot_2 and the $0°$ flags leaves $-\text{rot}_1 - \text{rot}_2 + 1 - x_f - x_{f'}$ possible combinations (or 0 if this value is negative) to set the shared rotations when

the result must obey the properties of a Kandinsky representation. In Figure 4.19c, $\text{rot}_1 = -1$ and $\text{rot}_2 = -1$ holds. The 0° flags allow for a 0° angle in f but not in f', thus $x_f = 0$ and $x_{f'} = 1$. This leaves only two ways to fix the shared rotations, namely $\text{rot}_f = 2$, $\text{rot}_{f'} = 0$ and $\text{rot}_f = 1$, $\text{rot}_{f'} = 1$. Counting those combinations for each of the 18 ways to fix the interface rotations and 0° flags of v in G_1 and G_2 (which can be done with a simple computer program) results in 330 combinations. Thus, the $18^{2k_{12}} \cdot 25^{k_{12}}$ combinations for the shared attachment vertices reduce to $330^{k_{12}}$. Hence, there are at most $18^{k_1+k_2} \cdot 330^{k_{12}}$ combinations for choosing attachment rotations, 0° flags, and shared rotations. Note that $18^2 = 324 \leq 330$ and thus we get the following.

$$18^{k_1+k_2} \cdot 330^{k_{12}} \leq \sqrt{330}^{k_1+k_2} \cdot \sqrt{330}^{2k_{12}} = \sqrt{330}^{k_1+k_{12}+k_2+k_{12}} \leq \sqrt{330}^{2k} = 330^k$$

To conclude, we get at most 330^k possibilities to choose all attachment rotations, all 0° flags, and all shared rotations. Once those are chosen, at most $(2\rho+1)^{\lfloor 1.5k \rfloor - 1}$ ways to choose rotations of the interface paths remain. Note that it is easy to list these combinations efficiently (without considering unnecessary combinations). □

Let G be a glueable subgraph of H. The *cost* of an interface class is the minimum cost of the Kandinsky representations it contains (recall that an interface class is a set of Kandinsky representations that have the same interface). The *cost table* of G is a table containing the cost of each interface class of G.

Lemma 4.12. *Let $G = G_1 \sqcup G_2$ be a merging step of width k with maximum rotation ρ. Given the cost tables of G_1 and G_2, the cost table of G can be computed on $O(k \cdot (2\rho+1)^{\lfloor 1.5k \rfloor - 1} \cdot 330^k)$ time.*

Proof. Start with a cost table for G with cost ∞ for every interface class. We iterate over all $(2\rho+1)^{\lfloor 1.5k \rfloor - 1} \cdot 330^k$ compatible choices for the shared rotations and the interfaces of G_1 and G_2 (Lemma 4.11). Consider a fixed choice and let $[\mathcal{K}_1]$ and $[\mathcal{K}_2]$ be the chosen interface classes of G_1 and G_2 with cost c_1 and c_2. In $O(k)$ time we can compute the interface class $[\mathcal{K}]$ we get for G. If $c_1 + c_2$ is less than the current cost $[\mathcal{K}]$, we set it to $c_1 + c_2$.

Note that the cost c_1 and c_2 imply the existence of Kandinsky representations $\mathcal{K}_1' \in [\mathcal{K}_1]$ and $\mathcal{K}_1' \in [\mathcal{K}_2]$ with cost c_1 and c_2. These two Kandinsky representations can be merged (using the fixed shared rotations) to a Kandinsky representation $\mathcal{K}' \in [\mathcal{K}]$. This representation clearly has cost $c_1 + c_2$ and thus the cost of $[\mathcal{K}]$ is at most $c_1 + c_2$.

On the other hand, assume that there exists a Kandinsky representation $\mathcal{K}$ of G with cost c. Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be the restrictions of $\mathcal{K}$ to G_1 and G_2, respectively. Let further c_1 and c_2 be the costs of $\mathcal{K}_1$ and $\mathcal{K}_2$, respectively. Then $c = c_1 + c_2$ holds. Moreover, the costs of the equivalence classes $[\mathcal{K}_1]$ and $[\mathcal{K}_2]$ are $c_1' \leq c_1$ and $c_2' \leq c_2$, respectively. As $[\mathcal{K}_1]$ and $[\mathcal{K}_2]$ can be merged to $[\mathcal{K}]$, at some point we set the cost of $[\mathcal{K}]$ $c_1' + c_2' \leq c_1 + c_2 = c$. Thus, on one hand, the cost of each equivalence class $[\mathcal{K}]$ of

G is never set to something below its actual cost, and on the other hand it is at some point set to a value that is at most its actual cost. Hence, this procedure yields the cost table of G. □

4.3.3 The Algorithm

The previous three lemmas together with a dynamic program on a sphere cut decomposition (which is a special type of branch decomposition) yield the following theorem.

Theorem 4.6. *An optimal Kandinsky representation of a plane graph G can be computed in $O(n^3 + n \cdot k \cdot (2\rho+1)^{\lfloor 1.5k \rfloor - 1} \cdot 330^k)$ time, where k is the branch width and ρ the maximum rotation of G.*

Proof. Let H be the plane graph. If H contains a degree-1 vertex, we can attach a cycle of length 4 to it. Computing an optimal Kandinsky representation of the resulting graph and removing this cycle from it obviously gives an optimal Kandinsky representation of H. Thus, we can assume without loss of generality that H does not contain degree-1 vertices.

In planar graphs, a branch decomposition with minimum width can be computed in polynomial [ST94] and even $O(n^3)$ [GT08] time. Moreover, Dorn et al. [Dor+10, Theorem 1] show that one can compute a sphere cut decomposition of width k from a given branch decomposition of width k in $O(n^3)$ time, if G does not contain degree-1 vertices. Without defining sphere cut decomposition precisely, it is essentially a rooted binary tree $\mathcal{T}$ (every node has two children or is a leaf) with a bijection between the edges of H and the leaves of $\mathcal{T}$ such that the following property holds. For every node μ of $\mathcal{T}$, the edges of H corresponding to leaves that are ancestors of μ induce a glueable subgraph of H. Denote this subgraph by G_μ.

Clearly, this implies that for an inner node μ with children μ_1 and μ_2, we get a merging step $G_\mu = G_{\mu_1} \sqcup G_{\mu_2}$. We process the inner nodes of $\mathcal{T}$ bottom up to compute the cost table of G_μ for every node μ. If a child μ_i (for $i = 1, 2$) of μ is a leaf, it corresponds to a single edge for which the cost tables are trivially known. Otherwise, we already processed the child μ_i and thus know the cost table of G_{μ_i}. Hence, by Lemma 4.12, we can compute the cost table of G_μ in $O(k \cdot (2\rho+1)^{\lfloor 1.5k \rfloor - 1} \cdot 330^k)$ time. Doing this for every inner node of $\mathcal{T}$ gives the claimed running time, since $\mathcal{T}$ contains $O(n)$ inner nodes. Moreover, for the root τ, we have $G_\tau = H$. Thus, after processing the root τ, we know the cost of an optimal Kandinsky representation of H. To actually compute an optimal Kandinsky representation of H (and not only its cost) one simply has to track the interface classes that lead to the optimal solution through the dynamic program. □

We get the following bounds for the maximum rotation ρ of a graph.

Lemma 4.13. *Let G be a graph with Kandinsky representation $\mathcal{K}$. Let Δ_F be the maximum face degree of G and let ρ be the maximum absolute rotation of interface paths of glueable subgraphs of G. The following holds.*

- *$\rho \leq m + \Delta_F - 2$, if $\mathcal{K}$ is an optimal Kandinsky representation.*
- *$\rho \leq (b+1) \cdot \Delta_F - b - 2$, if $\mathcal{K}$ is a b-bend Kandinsky representation.*

Proof. First note that every interface path of a glueable subgraph is a subpath of a face of G. Thus, interface paths have length at most $\Delta_F - 1$. We show that the maximum rotation of a path of this length satisfies the claimed bounds. Proving that the absolute value of the minimum rotation also satisfies these bounds is symmetric.

Consider the case that $\mathcal{K}$ is an optimal Kandinsky representation. As G admits a 1-bend representation [FKK97], there exists a representation with m bends and an optimal representation $\mathcal{K}$ has at most m bends. Thus, the edges on the interface path have at most m bends contributing rotation at most m. An interface path of length (in terms of number of edges) at most $\Delta_F - 1$ has at most $\Delta_F - 2$ inner vertices. If the rotation of each inner vertex is at most 1 we get the claimed inequality $\rho \leq m + \Delta_F - 2$. Consider a vertex v with rotation 2. Due to Property (5), at least one of the two edges in the path incident to v must have rotation -1. Thus, we can account rotation 1 even for vertices with rotation 2, yielding $\rho \leq m + \Delta_F - 2$.

In case $\mathcal{K}$ is a b-bend Kandinsky representation, the rotation contributed by the edges is at most $b \cdot (\Delta_F - 1)$. Together with the $\Delta_F - 2$ upper bound for the vertices, this gives $\rho \leq b \cdot \Delta_F - b + \Delta_F - 2 = (b+1) \cdot \Delta_F - b - 2$. □

We get the following corollaries by plugging the bounds of Lemma 4.13 into Theorem 4.6, using that the branch width of series-parallel graphs is 2, and that the branch width of planar graphs is in $O(\sqrt{n})$ (in fact, the branch width of a planar graph is at most $2.122\sqrt{n}$ [FT06]).

Corollary 4.2. *Let G be a plane graph with maximum face-degree Δ_F, and branch width k. An optimal Kandinsky representation can be computed in $O(n^3 + n \cdot k \cdot (2m + 2\Delta_F - 3)^{\lfloor 1.5k \rfloor - 1} \cdot 330^k)$ time. An optimal b-bend Kandinsky representation can be computed in $O(n^3 + n \cdot k \cdot ((2b+2) \cdot \Delta_F - 2b - 3)^{\lfloor 1.5k \rfloor - 1} \cdot 330^k)$ time.*

Corollary 4.3. *For series-parallel graphs an optimal Kandinsky representation can be computed in $O(n^3)$ time.*

Corollary 4.4. *For plane graphs an optimal Kandinsky representation can be computed in $2^{O(\sqrt{n} \log n)}$ time.*

4.4 Conclusion

In this chapter we have shown that bend minimization in the Kandinsky model is NP-complete, thus answering a question that was open for almost two decades. The proof also extends to the case that every edge may have at most one bend and to the case that empty faces are allowed.

On the positive side, we gave an algorithm with running time $2^{O(k \log n)}$ for graphs of branch width k. In fact, the problem is FPT with respect to $k + b + \Delta_F$, where k is the branch width, b is the maximum number of bends on a single edge in the drawing and Δ_F is the size of the largest face in the planar embedding. For general planar graphs this gives a subexponential exact algorithm with running time $2^{O(\sqrt{n} \log n)}$.

We leave open the question whether the number of parameters used to obtain an FPT algorithm can be decreased. Is the problem $W[1]$-hard when parameterized by branch width only?

Part II

Constrained Planarity

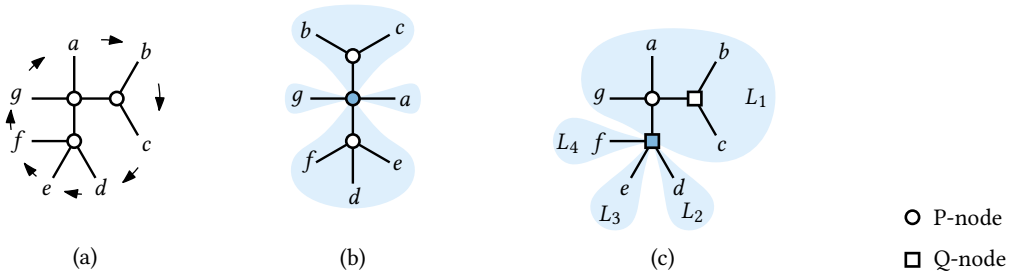
5 An Introduction to Simultaneous PQ-Ordering

The problem Simultaneous PQ-Ordering and its algorithmic solution for restricted cases [BR13] is a strong tool for solving constrained planarity problems. This chapter gives a short introduction to Simultaneous PQ-Ordering. It does not intend to provide all formal definitions or details on algorithmic solutions. It rather explains by means of multiple examples how to use the results on Simultaneous PQ-Ordering as a tool. It is not mandatory to read this chapter before the following chapters as the ingredients actually needed in Chapters 6 and 8 are formally defined in those chapters. However, they are certainly easier to understand after reading this chapter.

5.1 PQ-Trees Representing Cyclic Orders

Consider the tree T in Figure 5.1a. When walking around the embedded tree, we visit the leaves of T in a specific cyclic order, namely $[a,b,v,d,e,f,g]$. When changing the edge orderings of inner nodes, we get a different cyclic order. For example we get the order $[a,f,e,d,g,b,c]$ for the same tree T in Figure 5.1b by changing the edge ordering for two inner nodes. However, we cannot get every cyclic order. E.g., let μ be the marked inner node in Figure 5.1b. Then μ separates the leaves of T into four subsets $\{a\}$, $\{b,c\}$, $\{d,e,f\}$, and $\{g\}$. By changing the edge ordering for μ, we can reorder these subsets arbitrarily, but the leaves of each subset remain consecutive. In fact, we can get any cyclic order for the leaves in which these subsets are consecutive. We say that T *represents* this set of cyclic orders.

So far, we allowed to choose an arbitrary edge ordering for every inner node of T. In a PQ-tree, we have two different types of inner nodes, namely *P-nodes* and *Q-nodes*.



(a) (b) (c)

Figure 5.1: (a) A tree inducing a cyclic order on the leaves. (b) The same tree inducing a different order. (c) A PQ-tree.

For each P-node, we allow to choose an arbitrary edge ordering (as for all nodes in the above example). For Q-nodes, we assume the edge ordering to be fixed up to reversal, i.e., we can only choose its *orientation*. Figure 5.1c shows a PQ-tree with one P-node and two Q-nodes. Note that the PQ-tree in Figure 5.1c is more restrictive than the one in Figure 5.1a. E.g., the tree from Figure 5.1a allows an order of the leaves that contains $[a, e, d, f]$ as suborder (see Figure 5.1b). This is not possible for the tree in Figure 5.1c, as the marked Q-node forces these four leaves to have the order $[a, d, e, f]$ or its reversal $[a, f, e, d]$.

Let T be the PQ-tree form Figure 5.1c. How do the inner nodes of T exactly restrict the possible cyclic orders we can get for the leaves? As before, the P-node yields the four subsets $\{a\}$, $\{b, c\}$, $\{d, e, f\}$, and $\{g\}$, each of which must be consecutive. Consider the marked Q-node μ. It also partitions the leaves into subsets, namely $L_1 = \{a, b, c, g\}$, $L_2 = \{d\}$, $L_3 = \{e\}$, and $L_4 = \{f\}$. Again, each of the subsets must appear consecutively. Moreover, as μ is a Q-node, the subsets must have the cyclic order $[L_1, L_2, L_3, L_4]$ or its reversal $[L_1, L_4, L_3, L_2]$. Note that requiring L_1 and L_2 to be next to each other in the cyclic ordering is equivalent to requiring $L_1 \cup L_2$ to be consecutive. Thus, the restriction to the cyclic order $[L_1, L_2, L_3, L_4]$ or its reversal is equivalent to requiring that each of the leaf sets $L_1 \cup L_2$, $L_2 \cup L_3$, $L_3 \cup L_4$, and $L_4 \cup L_1$ appears consecutive.

It follows that the a cyclic order of the leaves is represented by T if and only if certain subsets appear consecutively in this order. Conversely, if we have a family $\{L_1, \ldots, L_k\}$ of subsets of the leaves, then there exists a PQ-tree that represents all cyclic orders in which L_i appears consecutive for each $i \in \{1, \ldots, k\}$ [BL76] (excluding the case where no such order exists).

A set of orders is *PQ-representable* if there exists a PQ-tree representing this set of orders.

5.2 PQ-Tree Reduction

Consider the PQ-tree T in Figure 5.2a. Assume we want to represent all cyclic orders that are represented by T with the additional requirement that the set $\{a, g\}$ is consecutive. This set of orders can again be represented by a PQ-tree and we denote this PQ-tree by $T + \{a, g\}$; see Figure 5.2a. The tree $T + \{a, g\}$ is the *reduction* of T with respect to $\{a, g\}$. We use the notation $T + \{a, g\}$ to represent the reduction for the following reason. As mentioned above, the PQ-tree T represents all orders in which certain subsets of the leaves appear consecutive. Thus, reducing T with $\{a, g\}$ simply adds $\{a, g\}$ to this family of subsets.

In the reduction in Figure 5.2a, most parts of T remain unchanged. In fact, only the P-node is split into two smaller P-nodes. When a reduction changes something for a Q-node of the PQ-tree, then the behaviour is somewhat reversed. Reducing T

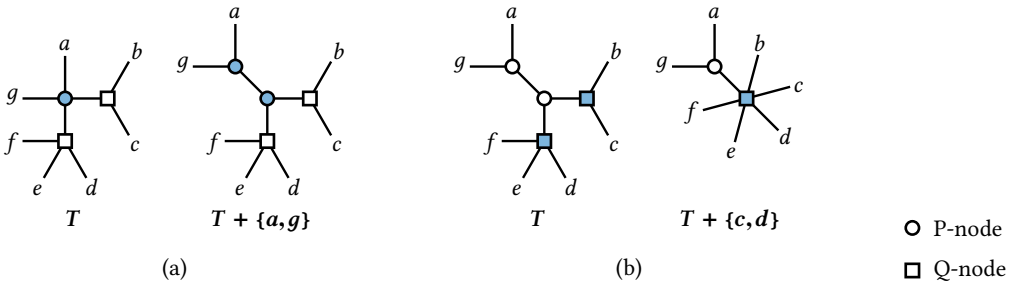


Figure 5.2: (a) A PQ-tree and its reduction with $\{a,g\}$. (b) A PQ-tree and its reduction with $\{c,d\}$.

in Figure 5.2b with the set $\{c,d\}$ merges the two Q-nodes (together with one P-node) into a single Q-node. Note that this on the first sight contradicting behaviour makes sense for the following reason. The PQ-tree consisting of a single large Q-node is the most restrictive PQ-tree we can get as it only represents a single cyclic order on the leaves (and its reversal). Thus, merging nodes into larger Q-nodes results in stronger restrictions. On the other hand, a P-node is the least restrictive PQ-tree as it allows all cyclic orders. Thus, splitting a P-node into multiple nodes (P- or Q-nodes) also results in stronger restrictions.

To conclude, a reduction of T can be obtained from T by splitting P-nodes and merging several nodes into a single Q-node. This leads to two simple definitions for a PQ-tree T and its reduction $T + S$ with a set of leaves S. Every P-node μ_S of $T + S$ was either already a P-node μ in T or is obtained by splitting a P-node μ in T. We say that μ_S *stems* from μ. The two marked P-nodes of $T + \{a,g\}$ in Figure 5.2a stem from the marked P-node of T. Moreover, for every Q-node μ of T, we find a Q-node μ_S in $T + S$ such that μ_S was obtained by merging μ with (maybe zero) other nodes. We say that μ_S is the *representative* of μ in $T + S$. The marked Q-node of $T + \{c,d\}$ in Figure 5.2b is the representative of both marked Q-nodes of T.

5.3 PQ-Tree Projection

Consider the PQ-tree T in Figure 5.3a with the leaves $a,\dots,g$. Assume we are actually only interested in the subset of leaves $\{a,c,e,g\}$ and how these leaves are ordered in orders represented by T. We get a PQ-tree representing exactly these orders by removing the remaining laves; see Figure 5.3a. We call the resulting PQ-tree the *projection* of T to $\{a,c,e,g\}$. We denote the projection of T to a subset S by $T|_S$.

Assume we want to project T to the leaves $\{a,b,c,d\}$ instead. Removing all other leaves results in a tree with a P-node of degree 2; see the colored edges in Figure 5.3b. Note that this P-node is superfluous as it has a unique edge ordering. We obtain the projection $T|_{\{a,b,c,d\}}$ shown in Figure 5.3b.

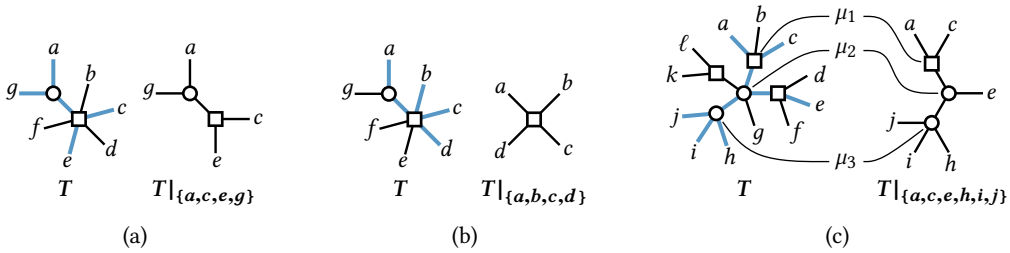
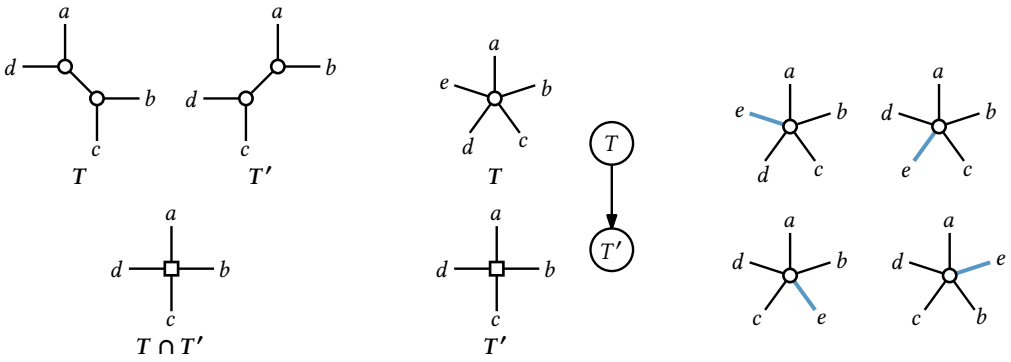


Figure 5.3: (a) The PQ-tree T and its projection to $A = \{a, c, e, g\}$. (b) The projection of T to $B = \{a, b, c, d\}$. (d) A larger PQ-tree T with its projection to $C = \{a, c, e, h, i, j\}$.

Let $A = \{a, c, e, g\}$ and $B = \{a, b, c, d\}$ be the leaf sets from Figure 5.3a and Figure 5.3b, respectively. The two projections $T|_A$ and $T|_B$ are different in the sense that every inner node of T is still contained in $T|_A$, whereas the P-node of T disappeared in $T|_B$. This has the following effect. Assume we already fixed the order $[a, c, e, g]$ for A (as in the embedding of $T|_A$ in Figure 5.3a). The only order represented by T that contains $[a, c, e, g]$ as suborder is $[a, b, c, d, e, f, g]$ (as in the embedding of T in Figure 5.3a). In other words, fixing the edge ordering of the P-node in $T|_A$ determines the edge ordering of the P-node in T and fixing the orientation of the Q-node in $T|_A$ determines the orientation of the Q-node in T.

Now consider $T|_B$. Fixing the orientation of the Q-node of $T|_B$ determines the orientation of the Q-node in T. However, the P-node of T does not exist in $T|_B$. Thus, we are free to choose an arbitrary edge ordering for this P-node. In terms of orders, we get two orders represented by T that extend the given order $[a, b, c, d]$. These two orders are $[a, b, c, d, e, f, g]$ and $[a, g, b, c, d, e, f]$. We say that the Q-node of T is *fixed* by $T|_B$, whereas the P-node is *free*.

Let us consider the larger example in Figure 5.3c. We projected the PQ-tree T to the leaves $C = \{a, c, e, h, i, j\}$. The Q-node μ_1 and the P-nodes μ_2 and μ_3 are contained in T and $T|_C$, while the two other Q-nodes of T disappeared in $T|_C$. These two Q-nodes are free as choosing their orientations is completely independent from ordering the leaves in C. The Q-node μ_1 is fixed as fixing an order for C determines the orientation of μ_1. Similarly, the P-node μ_3 is fixed as fixing an order for C completely determines the edge ordering for μ_3. The P-node μ_2 is only partially fixed as fixing an order for C determines only the order of three edges around μ_2. The two other edges incident to μ_2 can be added arbitrarily to this order. However, we do not distinguish between P-nodes being partially or completely fixed. Thus, in this example, the nodes μ_1, μ_2, and μ_3 are fixed, and the two remaining Q-nodes are free.



(a) (b) (c)

Figure 5.4: (a) Two different PQ-trees with the same leaf set and their intersection. (b) Two PQ-trees such that the leaves of T include the leaves of T'. (c) All orders (up to reversal) represented by T and T' in (b).

5.4 Simultaneously Ordering Two PQ-Trees

In the previous section, we already considered multiple PQ-trees at the same time, namely a tree T together with its projection $T|_S$ to a subset S of its leaves. We were then interested in simultaneously embedding T and $T|_S$ such that the resulting leaf orders fit to each other, i.e., the order for T is an extension of the order for $T|_S$. Now we consider the more general situation that T and T' are PQ-trees with leaf sets L and L' that share some leaves, i.e., $L \cap L' \neq \emptyset$.

The example in Figure 5.4a shows the simplest case where $L = L'$. The two P-nodes in T require the leaf sets $\{a, d\}$ and $\{b, c\}$ to be consecutive. The tree T' requires $\{a, b\}$ and $\{c, d\}$ to be consecutive. If we want to satisfy the constraints of both PQ-trees at the same time, we essentially want all four sets to be consecutive. This results in a new PQ-tree, which we denote by $T \cap T'$; see Figure 5.4a. In general, there is either no order represented by T and by T' or we can find such a PQ-tree $T \cap T'$. In the former case one can formally say that $T \cap T'$ is the *null tree*, representing the empty set of orders. To conclude, every set of orders that can be represented by two PQ-trees with the same leaf set can also be represented by a single PQ-tree.

Now consider the trees T and T' with $L' \subseteq L$ in Figure 5.4b. Actually, we only require that we have an injective map from L' to L. To keep it simpler, we work with the stricter requirement $L' \subseteq L$ where possible. We say that T' and T are in a *child-parent relation*, where T is the parent and T' is the child. We also interpret this kind of relation as an arc from T to T'. The child T' requires the leaves a, b, c, and d to have the order $[a, b, c, d]$ or its reversal $[a, d, c, b]$. As the parent T consists of a single P-node, the leaf e (which is only contained in T) can be added to this order arbitrarily. Thus,

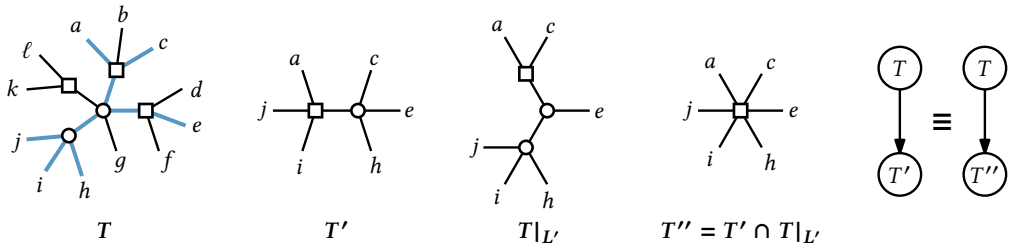


Figure 5.5: Two PQ-trees T and T', the reduction of T to the leaves L' of T', and the intersection of T' with this projection.

the trees T and T' represent together the orders $[a,b,c,d,e]$, $[a,b,c,e,d]$, $[a,b,e,c,d]$, $[a,e,b,c,d]$, and the reversal of these four orders; see also Figure 5.4c. This set of eight orders cannot be represented by a single PQ-tree for the following reason. Consider for example the subset of leaves $S = \{a,b\}$. Then S is not consecutive in the order $[a,e,b,c,d]$. The same holds for every other set S of size 2 or 3. Clearly, if S has size 1 or size 4, then S is trivially consecutive in any leaf order. Thus, the set of orders simultaneously represented by T and T' cannot be expressed in terms of consecutivity constraints, which makes it impossible to represent it with a single PQ-tree. Hence, the combination of two PQ-trees is more powerful than a single PQ-tree.

Note that finding an order in the previous example that is represented by T and T' is easy. One can simply take an arbitrary order represented by the child T' and extend it to an order represented by the parent T. This is not true in general. Let T and T' be the PQ-trees in Figure 5.5. Clearly, there is an order represented by T and T'; in Figure 5.5, the order for T extends the order for T'. However, not every order of the leaves L' represented by the child T' can be extended to an order of the larger leaf set L represented by the parent T. Exchanging for example c and e is possible for T' but not for T without also changing the order of a and e. Can we somehow systematically find an order for the child T' that can be extended to an order of the parent T, or decide that no such order exists?

To answer this question, it would be helpful to know which orders of L' can be extended to an order of L that is represented by T. Fortunately, we already know that this set of orders is represented by the projection $T|_{L'}$ of T to L'; see Figure 5.5. It then remains to choose an order of L' that is not only represented by T' but also by $T|_{L'}$, which can then be extended to an order of L represented by the parent T. Which orders are represented by T' and by $T|_{L'}$? Exactly those, that are represented by the PQ-tree $T'' = T' \cap T|_{L'}$; which is also shown in Figure 5.5. To conclude, an order is represented by T and T' if and only if it is represented by T and $T'' = T' \cap T|_{L'}$. If we then actually want to have an order represented by T and T', we can simple choose an arbitrary order for T'', which is also represented by T', and can be sure that it is extendable to an order represented by T. This process of replacing the child T'

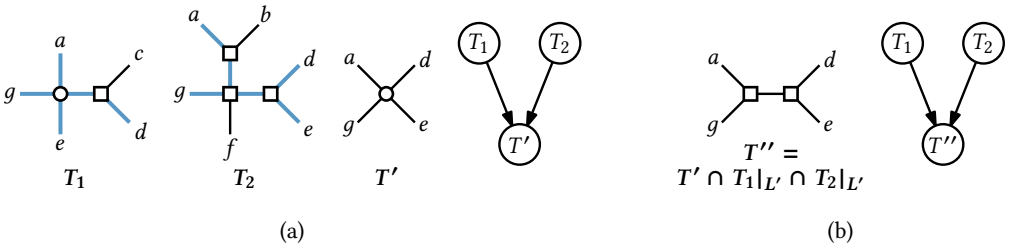


Figure 5.6: (a) Two PQ-trees T_1 and T_2 with the common child T'. (b) Normalizing both arcs results in the new instance where T' is replaced by T''.

with the more restrictive (but in this situation equivalent) PQ-tree T'' is also called *normalization* of the arc from the parent T to the child T'.

5.5 Simultaneously Ordering Multiple PQ-Trees

In the previous section, we considered two PQ-trees T and T' that are in a child-parent relation. Now we go a step further and consider multiple PQ-trees with multiple child-parent relations at the same time. Let T_1, T_2, and T' be the PQ-trees from Figure 5.6a. As the leaf set of T' is a subset of the leaves of T_1 and a subset of the leaves of T_2, we can use T' as a child of T_1 and as a child of T_2. Can we now *simultaneously* choose orders for all three PQ-trees, i.e., can we choose orders such that the orders chosen for T_1 and for T_2 both extend the order chosen for T'? The problem Simultaneous PQ-Ordering consists of answering this question.

Note that the common child T' in a sense synchronizes the order of the leaves $\{a,d,e,g\}$ in T_1 and T_2. Let us first focus on the arc from T_1 to T'. We learned from the previous section that we can first normalize the arc (by replacing T' with $T' \cap T_1|_{L'}$) without loosing solutions. Afterwards, we can choose an arbitrary order for the child and extend it to T_1. In the example from Figure 5.6a, the arc from T_1 to T' is already normalized. Of course the same arguments apply to the arc from T_2 to T'. Normalizing this arc yields the tree T'' in Figure 5.6b. Thus, having T' as a common child of T_1 and T_2 is equivalent to having T'' as a common child of T_1 and T_2. Moreover, finding simultaneous orders in the latter case is easy: Choose an arbitrary order for T'' and extend this order to both parents T_1 and T_2. Thus, normalizing every arc is sufficient to solve an instance of Simultaneous PQ-Ordering in which a single child has multiple parents.

In the next example, we reverse this situation and consider a single parent T having two children T_1 and T_2; see Figure 5.7a. Let L, L_1, and L_2 be the leaves of T, T_1, and T_2, respectively. The two arcs from T to T_1 and T_2 are already normalized. Thus, if we fix an order for T_1, it can always be extended to an order of T and the same holds for T_2. However, we have to make sure that these two orders extend to the same order of T. To

this end, we have to understand how the orders of T_1 and T_2 interact when extending them to an order in T. First focus on the arc from T to T_1. As it is normalized, T_1 can be obtained from T, by first projecting T to the leaves L_1 of T_1 and then applying some reductions. In this case, the projection $T|_{L_1}$ contains the nodes μ_1, μ_2, and μ_3 (but not μ_4). Recall that we called these nodes fixed, as fixing an order for the leaves L_1 already (partially) determines the choice at these nodes. In the same sense, the nodes μ_1, μ_2, and μ_3 are *fixed* with respect to T_1, while μ_4 is *free*.

Consider the Q-node μ_1 that is fixed with respect to T_1. Recall that Q-nodes are only merged (and never split) by applying reductions, yielding a unique representative in the resulting tree. In this example, the representative of μ_1 in T_1 is μ_5. Thus, choosing an orientation for μ_5 in T_1 determines the orientation of μ_1 in T. Note that μ_1 is also fixed with respect to the other child T_2, in which it has μ_7 as representative. Thus, to make sure that orders chosen for T_1 and for T_2 extend to the same order of T, it is necessary that the orientations of μ_5 and μ_7 imply the same orientation for μ_1. This can be ensured using equations and inequalities on Boolean variables that represent the orientation of Q-nodes; we call these constraints the *Q-constraints*. Thus, enforcing that the orders we choose for T_1 and T_2 have no contradicting implications on the orientation of Q-nodes in T is easy.

Consider the P-node μ_3 that is fixed with respect to T_1, i.e., choosing an order for T_1 fixes the order of the four edges incident to μ_3. Actually, the leaves b, e, h, and i of T_1 are *representatives* of the top, right, bottom, and left edge incident to μ_3, respectively. I.e., choosing an order for these four leaves already determines the edge ordering of μ_3. Similarly, the leaves b, e, g, and i are representatives in the tree T_2. If we have for example the leaf order $[b,e,h,i]$ for T_1, then T_2 must have the leaf order $[b,e,g,i]$; otherwise, we get conflicting implications on the edge ordering of the P-node μ_3. Thus, we want to ensure that some leaves are ordered the same in T_1 and in T_2. Recall that this is (more or less) exactly what a common child of T_1 and T_2 ensures. Thus, we create a new PQ-tree T' consisting of a single P-node with leaf set $L' = \{b,e,g/h,i\}$ (the element g/h is a single leaf). We add T' to the instance of Simultaneous PQ-Ordering as a common child of T_1 and T_2. Note that L' is formally not a subset of L_1 and L_2. However, we have the (natural) injective maps, mapping b, e, and i to themselves and g/h to h and g in T_1 and T_2, respectively. Figure 5.7b shows the tree T' and the instance of Simultaneous PQ-Ordering.

If we now have orders for T_1 and T_2 that both extend the same order of T', then we know that T_1 and T_2 imply the same edge ordering of μ_3 in T. We already know that we can get orders of T_1 and T_2 that both extend an order of a common child by first normalizing the two corresponding arcs, then choosing an arbitrary order for the child, and finally extending it to T_1 and T_2. The tree we obtain after normalizing both arcs is T'' shown in Figure 5.7c. Note that the Q-nodes μ_5 and μ_6 are fixed by the child T''. As before, we synchronize their orientation using Q-constraints.

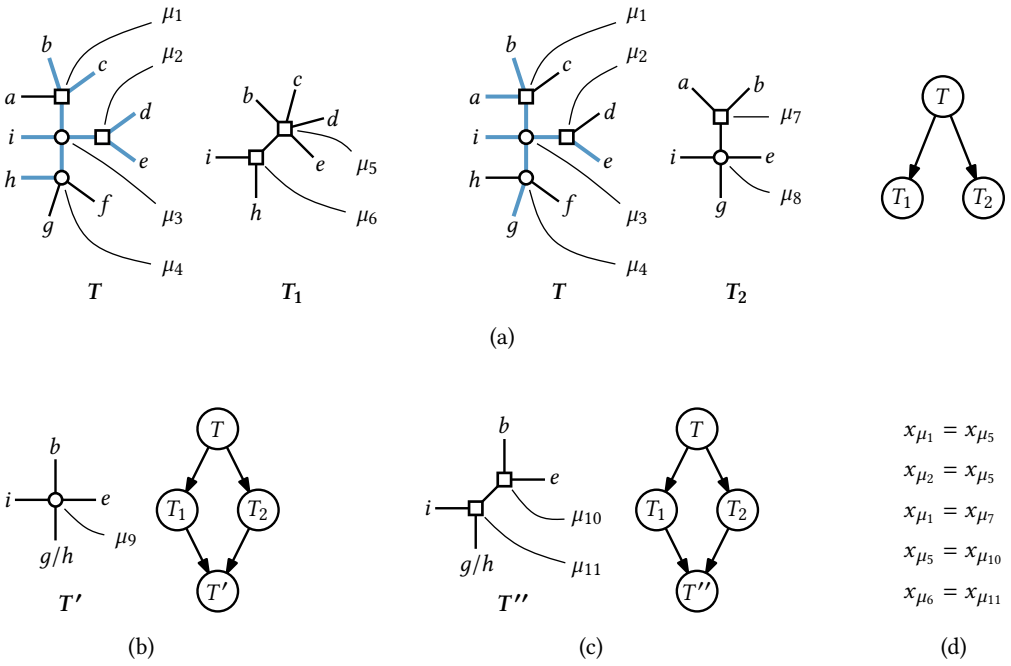


Figure 5.7: (a) The PQ-tree T with the two children T_1 and T_2. For better comparability with the two children, the parent T is drawn twice. (b) Adding a common child T' to T_1 and T_2 to make sure they imply the same edge ordering for μ_3. (c) Normalized Instance. (d) The Q-constraints.

To recap, consider the resulting instance of Simultaneous PQ-Ordering in Figure 5.7c, where T has the two children T_1 and T_2 that have T'' as a common child. Figure 5.7d shows the Q-constraints we get for this instance; x_μ represents the orientation of μ, where $x_\mu = 0$ corresponds to the orientation shown in Figure 5.7 and $x_\mu = 1$ to its reversal. To get a solution for this instance, we first choose an order for T''; in this example, the order $[b, e, g/h, i]$ was chosen. As the arcs from the two parents were normalized, we know that we can extend this order to orders of T_1 and of T_2. It remains to choose an order for T, which we do by looking at one inner node of T after another. First consider the node μ_1. It is a Q-node that is fixed by T_1 and T_2 with the representatives μ_5 and μ_7. The Q-constraints make sure that they imply the same orientation for μ_1 and we simply choose this orientation. For μ_2, it is even simpler, as μ_2 is only fixed by T_1. Thus, we can simply orient μ_2 according to its representative μ_5 in T_1. The P-node μ_4 is neither fixed by T_1 nor by T_2. Thus, we can actually choose an arbitrary edge ordering for μ_4. Finally, μ_3 is fixed with respect to both trees. Thus, T_1 and T_2 imply an edge ordering for μ_3. However, they imply the same edge ordering as they both extend the order chosen for T'', whose leaves are in one-to-one correspondence with the edges incident to μ_3.

In the example in Figure 5.7, we had to add one tree T'', also called *expansion tree* as a common child of T_1 and T_2, as T had one P-node that was fixed with respect to T_1 and T_2. If T has multiple P-nodes that are fixed with respect to both children, we have to add several expansion trees, one for each commonly fixed P-node.

In the case where we had multiple parents with a single common child, the case of two parents directly extended to the case of an arbitrary number of parents. In the case of a single parent with multiple children, this is no longer true for the following reason. Assume the parent T has a P-node μ that is fixed with respect to multiple children, each child fixing the order of an arbitrary subset of edges incident to μ. Formulating constraints on the orders of these subsets such that all these orders can be extended to an edge ordering of μ is not as simple as in the case with two children fixing μ. In fact, deciding for given cyclic orders on subsets whether they can be extended simultaneously to an order of the whole set is NP-hard [GM77]. However, the approach presented above still works if T has multiple children, as long as every P-node of T is fixed with respect to at most two children.

5.6 Solvable Instances

In the previous section, we already saw some simple instances of Simultaneous PQ-Ordering together with algorithmic approaches to solve them that actually run in polynomial time. In general, an instance of Simultaneous PQ-Ordering can be an arbitrary DAG (with a PQ-tree associated with every node) and the problem is NP-hard. However, the strategies developed above can be used to solve a significant set of instances.

We can summarize the overall strategy as follows. First, every arc is normalized to make sure that an order chosen for the child can always be extended to an order of the parent. To make sure that the Q-nodes of all trees are consistent, we use a set of equations and inequalities on Boolean variables, the Q-constraints. If the instance contains a tree T such that T has a P-node μ that is fixed with respect to two children T_1 and T_2, we expand the instance by adding a common child, the expansion tree, to T_1 and T_2. After this process of adding expansion trees terminates (which it does if some special cases are handled carefully), one can choose orders bottom up in the resulting DAG.

There are two things to note. First, we are not able to solve an instance if we encounter a P-node that is fixed with respect to three or more children during the expansion process. Second, adding expansion trees may lead to additional P-nodes that are fixed with respect to two or in the worst case even more children. In the following we give a simple criterion for an instance of Simultaneous PQ-Ordering that guarantees that the expansion terminates without encountering a P-node that is

fixed with respect to three ore children. For these instances, the above strategy yields an algorithm solving Simultaneous PQ-Ordering in quadratic time.

Let D be an instance of Simultaneous PQ-Ordering (e.g., the one in Figure 5.8, which is explained in detail in a moment) and let T be a PQ-tree that is a source of D (i.e., T has no parents). For a P-node μ of T, we say that its *fixedness* $\text{fixed}(\mu)$ is the number of children of T fixing μ. During the expansion, we never add new children to sources in D. Thus, $\text{fixed}(\mu) \leq 2$ makes sure that μ will never be fixed with respect to three or more children of T. For non-source trees, we have to be more careful. Let T be a tree with parents $T_1, \ldots, T_\ell$ and let μ be a P-node of T. Recall that T can be obtained form T_i (for $i = 1, \ldots, \ell$) by projecting T_i to the leaves of T and applying some reductions. Applying reductions can split P-nodes into several P- or Q-nodes, but it can never create new or larger P-nodes from something that was not a P-node in T_i. Thus, the P-node μ of T stems from a unique (maybe larger) P-node μ_i of T_i. One can see that a fixedness of 2 for μ_i can be responsible for adding one child to T that fixes μ. This motivates the following definition of the fixedness for μ, where k is the number of children that T already has in D that fix μ.

$$\text{fixed}(\mu) = k + \sum_{i=1}^{\ell} (\text{fixed}(\mu_i) - 1) \tag{5.1}$$

The instance D of Simultaneous PQ-Ordering is *k-fixed*, if the fixedness of every P-node in every PQ-tree is at most k.

Theorem 5.1 ([BR13]). *Simultaneous PQ-Ordering can be solved in quadratic time for 2-fixed instances.*

Now consider the example in Figure 5.8. The trees T_1 and T_2 are the sources of D. Consider the node μ_1 in T_1. It is fixed by the child T_3 and by the child T_4. However, it is not fixed by the third child T_5, as the projection of T_1 to the leaf set $\{a, b, c, d, e, f\}$ of T_5 does not contain μ_1. Thus, μ_1 is fixed by two children, i.e., $\text{fixed}(\mu_1) = 2$. The second P-node μ_2 of T_1 is not fixed by T_3 but it is fixed by T_4 and by T_5. Thus, the fixedness of μ_2 is also 2. The P-node μ_3 in the other source T_2 is fixed by both children T_5 and T_6, thus we have $\text{fixed}(\mu_3) = 2$. The P-node μ_4 is not fixed with respect to T_6 as the projection of T_2 to the leaves of T_6 does not contain μ_4. Thus, we have $\text{fixed}(\mu_4) = 1$.

Continue with the non-source tree T_4 and first consider the P-node μ_5. First note that T_4 has no children that could fix μ_5. Thus, the k in Equation (5.1) is 0. We next have to figure out from which P-node of the parent T_1 the node μ_5 stems. Note that T_4 is simply the projection of T_1 to the leaves of T_4. It is thus not hard to see that μ_5 stems from μ_1. As this is the only parent of μ_5 we get $\text{fixed}(\mu_5) = k + (\text{fixed}(\mu_1) - 1) = 1$. Similarly, the P-node μ_6 stems from μ_2 in T_1, which has also fixedness 2, resulting in $\text{fixed}(\mu_6) = 1$. For T_5 it gets more interesting. First consider the P-node μ_7. The child T_7 does not fix μ_7, thus $k = 0$. Note that projecting T_1 to the leaves of T_5 results in a single P-node

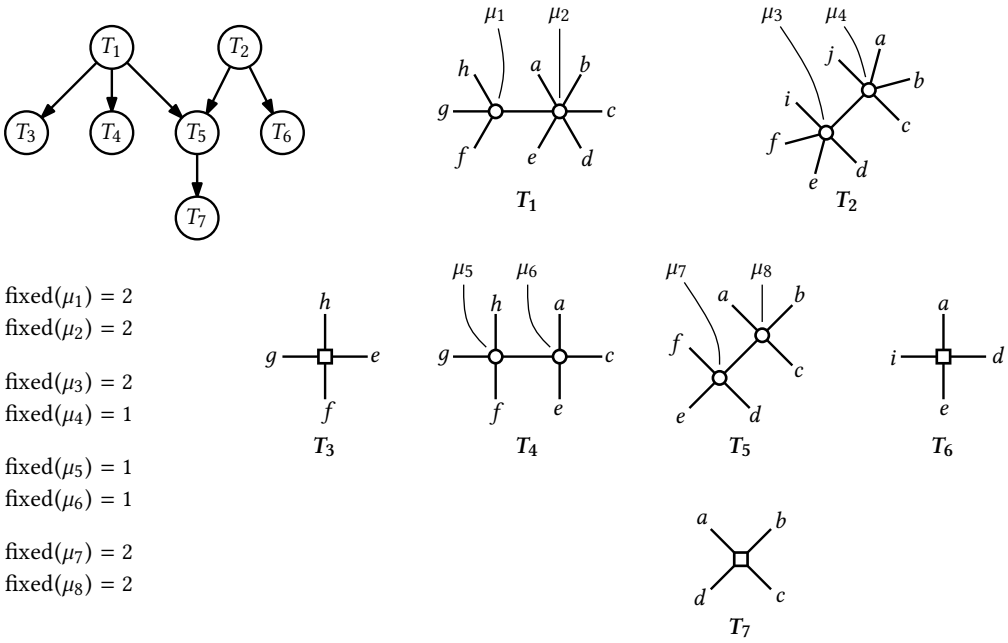


Figure 5.8: A 2-fixed instance D of Simultaneous PQ-Ordering.

stemming from μ_2. To obtain T_5, this P-node is split into two P-nodes. Thus, both P-nodes μ_7 and μ_8 stem from μ_2. Considering the other parent T_2, the node μ_7 stems from the node μ_3. Thus, we get $\text{fixed}(\mu_7) = k + (\text{fixed}(\mu_2) - 1) + (\text{fixed}(\mu_3) - 1) = 0 + 1 + 1 = 2$. The other P-node μ_8 is fixed by the child T_7, thus we have $k = 1$ in this case. As mentioned before μ_8 stems from μ_2 in T_1. Moreover, it stems from μ_4 in T_2. Thus, we get $\text{fixed}(\mu_8) = k + (\text{fixed}(\mu_2) - 1) + (\text{fixed}(\mu_4) - 1) = 1 + 1 + 0 = 2$. As the fixedness of no P-node exceeds 2, the instance in Figure 5.8 is 2-fixed and can be solved using Theorem 5.1.

5.7 Representing Planar Embeddings

Consider the biconnected planar graph G in Figure 5.9a. Our goal is to represent the planar embeddings of G (on the sphere) using an instance of Simultaneous PQ-Ordering. Consider the vertex v with the incident edges a, b, c, d, e, and f. Which edge orderings for v can we get in planar embeddings of G? We claim that all possible edge orderings are represented by the PQ-tree T_v in Figure 5.9b having one leaf for each edge incident to v. First note that u and v are a separating pair partitioning the edges incident to v into the subsets $\{a, b\}$, $\{c\}$, $\{d, e\}$, and $\{f\}$. We can reorder these subsets arbitrarily but each subset has to be consecutive. The P-node in T_v models exactly this behaviour. The split component with respect to $\{u, v\}$ that contains the

edges a and b can be either embedded as in Figure 5.9a, or its embedding can be flipped. This binary decision corresponds to choosing an orientation for one of the Q-nodes in T_v (the unmarked Q-node in Figure 5.9b). Similarly, the marked Q-node corresponds to the decision of flipping the split component containing the edges d and e. Thus, the possible edge orderings at v are represented by the PQ-tree T_v. We call T_v the *embedding tree* of v.

Conversely, for any PQ-tree T, one can build a planar graph such that a vertex v has T as embedding tree (e.g., [Len89]). However, in this section, we are more interested in representing planar embeddings using PQ-trees than in simulating PQ-trees using planar graphs.

As for the vertex v, we get the embedding tree T_u for the vertex u. As before, reordering the edges incident to the P-node of T_u corresponds to reordering the four split components with respect to $\{u, v\}$.

Consider the SPQR-tree $\mathcal{T}$ of G; see Figure 5.9c. The embedding choice of reordering the split components with respect to the separating pair $\{u, v\}$ is represented by the P-node μ_1 of $\mathcal{T}$. Thus, the order we choose for the virtual edges when embedding $\text{skel}(\mu_1)$ determines the orders of the P-nodes in the PQ-trees T_u and T_v. Similarly, choosing one of the two embeddings for the skeleton of the R-node μ_2 of $\mathcal{T}$ determines the orientation of the marked Q-nodes of T_u and T_v. The orientation of the other Q-node of T_v is determined by the embedding of the skeleton of the R-node μ_3 in the SPQR-tree $\mathcal{T}$.

To conclude, the embedding tree T_v of v, which is a PQ-tree representing all edge orderings of v, has a P-node for every P-node of the SPQR-tree $\mathcal{T}$ for which v is a vertex of the skeleton. Similarly, T_v has a Q-node for every R-node of $\mathcal{T}$ for which v is a vertex of the skeleton. The fact that P-nodes of the PQ-trees correspond to P-nodes of the SPQR-trees is by coincidence and one should not be confused by the fact that Q-nodes of PQ-trees correspond to R-nodes of the SPQR-tree (and not to its Q-nodes).

For a biconnected planar graph G, we now know that every vertex v has an embedding tree T_v, which is a PQ-tree representing exactly the edge orderings of v that we can get in a planar embedding of G. However, this does not suffice to represent planar embeddings of G as the choices for different embedding trees are not independent. Consider again the example in Figure 5.9. Choosing an orientation for the marked Q-node of T_v determines the embedding of $\text{skel}(\mu_2)$ in the SPQR-tree. Similarly, choosing an orientation for the Q-node of T_u also determines the embedding of $\text{skel}(\mu_2)$. Thus, we have to make sure that these two choices are consistent, i.e., they imply the same embedding for $\text{skel}(\mu_2)$. This can be done by adding a common child T_{μ_2} to T_v and T_u with three leaves that are identified with d, e, and f in T_v and with k, j, and i in T_u; see Figure 5.9d.

Similarly, the edge orderings of the P-nodes in T_v and T_u both determine the order of virtual edges in $\text{skel}(\mu_1)$. For T_v we can choose the leaves a, c, d, and f to represent

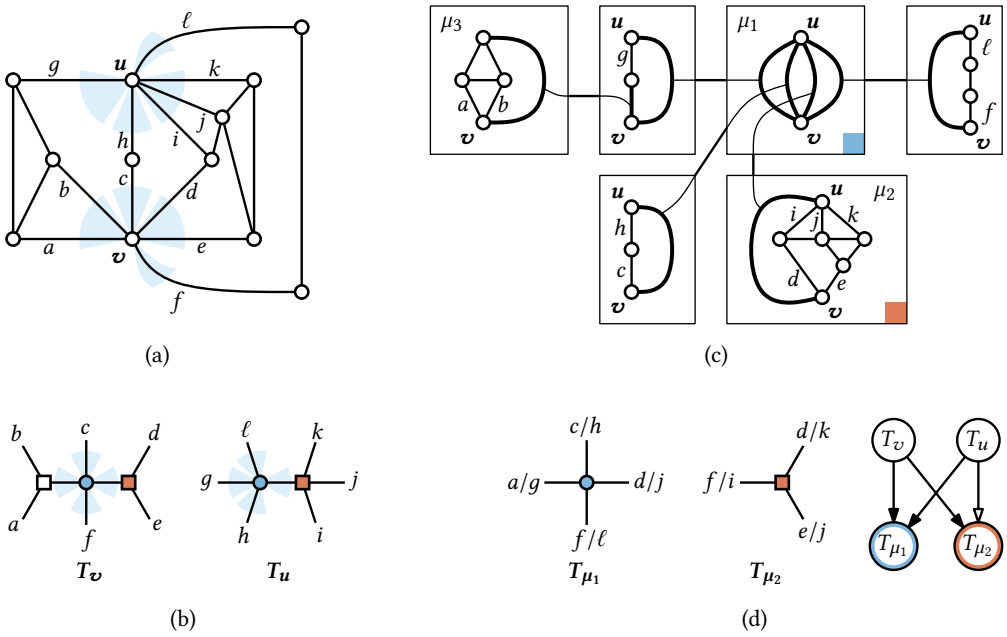


Figure 5.9: (a) A biconnected planar graph G. (b) The embedding trees T_v and T_u describing the possible edge orderings of u and v in G. (c) The SPQR-tree of G. The embedding decisions of the P-node μ_1 and the R-node μ_2 correspond to the ordering and orientation decisions of the P-nodes and marked Q-nodes, respectively, in the embedding trees T_v and T_u. (d) Adding children to T_u and T_v to enforce consistent edge orderings.

this order and for T_u this order can be represented by the leaves g, h, j, and ℓ. Note that a and g correspond to each other as they both belong to the same split component. Similarly, c and h represent the second split component, d and j the third, and f and ℓ the fourth. As for the Q-node, we can ensure that these representatives for the split components are ordered the same with respect to both embedding trees by adding a PQ-tree T_{μ_1} as common child of T_u and T_v. These trees T_{μ_1} and T_{μ_2} are called *consistency trees*.

The consistency tree T_{μ_1} for the P-node μ_1 does actually not exactly what we want it to do for the following reason. It ensures that the cyclic order of split components with respect to $\{u, v\}$ around v is the same as the cyclic order of these split components around u. However, in a planar embedding of G, the cyclic order of split components around v is the reversal of the cyclic order of split components around u. Thus, we actually do not want to ensure that these orders are the same but that one is the reversal of the other. We can achieve this by slightly extending the problem Simultaneous PQ-Ordering. As defined in the previous section, the arc from T_v to T_{μ_1} requires the order chosen for T_v to be an extension of the order chosen for T_{μ_1}. For the arc from T_u to T_{μ_1}, we instead require that the order chosen for T_u is an extension of the reversal of

the order chosen for T_{μ_1}. We say that this arc is an *reversing arc* (note that the reversing arc is illustrated differently in Figure 5.9d). The algorithm solving Simultaneous PQ-Ordering for 2-fixed instances still works in the presence of reversing arcs [BR13].

To conclude, the instance of Simultaneous PQ-Ordering in Figure 5.9d includes the embedding trees T_u and T_v that describe all possible edge orderings for the nodes u and v, respectively. Moreover, when choosing orders for both trees at the same time (i.e., finding a solution of the Simultaneous PQ-Ordering instance), the consistency trees ensure that G admits a planar embedding realizing the chosen edge orderings for both nodes u and v at the same time. Clearly, we can extend this instance to have an embedding tree for every vertex of the graph. The resulting instance is called the *PQ-embedding representation* of G. Every solution to this PQ-embedding representation fixes an edge ordering for every vertex and the consistency trees make sure that these edge orderings correspond to a planar embedding of G. Moreover, the edge orderings of every planar embedding yield a solution for the PQ-embedding representation. Thus the solutions of the PQ-embedding representation are in one-to-one correspondence with the planar embeddings of the biconnected graph G.

With the PQ-embedding representation, it is easy to restrict the edge orderings in planar embeddings by applying these restrictions to the embedding trees. Consider for example the situation where we have two biconnected planar graphs $G^{①}$ and $G^{②}$ that share a common subgraph. Assume we want planar embeddings of $G^{①}$ and $G^{②}$ such that the order of the common edges around a vertex v is the same in both embeddings. Then the PQ-embedding representations of $G^{①}$ and $G^{②}$ both include an embedding tree for v. Let $T_v^{①}$ and $T_v^{②}$ be these embedding trees. We have to enforce that the leaves shared by $T_v^{①}$ and $T_v^{②}$ (representing the common edges incident to v) are ordered the same, which can be done by adding a PQ-tree as common child to $T_v^{①}$ and $T_v^{②}$. This leads to a polynomial-time algorithm for the simultaneous embedding problem Sefe when both graphs are biconnected with a connected common graph.

The final consideration we want to make in this chapter concerns the fixedness of the PQ-embedding representation. Recall, that we can only solve 2-fixed instances of Simultaneous PQ-Ordering efficiently. Consider the embedding tree T_v in Figure 5.9b. Its P-node corresponds to the P-node μ_1 of the SPQR-tree and thus it is fixed by the child T_{μ_1}. However, the P-node is not fixed by any other child. Thus, its fixedness is 1. The same holds for the P-node of T_u (as for each P-node in every embedding tree). The consistency tree T_{μ_1} consists of a single P-node. It has no children (and thus no children fixing its P-node). Moreover, as $\text{skel}(\mu_1)$ has only two vertices (as every P-node skeleton has), T_u and T_v are the only parents of T_{μ_1}. For both parents, the P-node of T_{μ_1} stems from a P-node with fixedness 1. Thus, the fixedness of T_{μ_1} is 0. Note that adding a child to the embedding tree T_v may increase the fixedness of a P-node in T_v to 2. Similarly, adding a child to T_u may increase the fixedness of its P-node to 2. For T_{μ_1} this also results in a fixedness of 2.

To conclude, the PQ-embedding representation is 1-fixed. It remains 2-fixed when adding one child to each embedding tree. In fact, one can add multiple children to the same embedding tree, as long as each P-node of this embedding tree is fixed by at most one of these additional children.

6 A New Perspective on Clustered Planarity

In this chapter, we consider the problem Clustered Planarity. We introduce the cd-tree data structure and give a new characterization of c-planarity based on the cd-tree. This characterization leads to efficient algorithms for Clustered Planarity in the following cases. (i) Every cluster and every co-cluster (complement of a cluster) has at most two connected components. (ii) Every cluster has at most five outgoing edges. For both cases, the computational complexity was unknown before.

Moreover, the cd-tree reveals interesting connections between c-planarity and planar embedding with constraints on the edge orderings. On one hand, this gives rise to a bunch of new constrained planarity problems related to Clustered Planarity. On the other hand it provides a new perspective on previous results, partially explaining why seemingly unnatural restrictions help to efficiently solve Clustered Planarity.

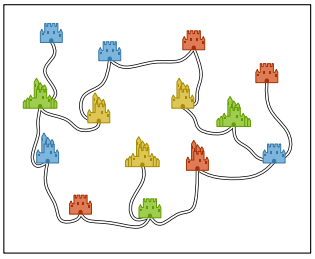
This chapter is based on joint work with Ignaz Rutter [BR14].

6.1 Introduction

We want to begin this chapter with a short story about a king stumbling over a variant of Clustered Planarity while drawing up his last will.

~ A King and his Sons ~

Once upon a time there was a powerful and wise king ruling a prosperous kingdom. He was in the best of health and his physician assured him a long-lasting life. However, in his wisdom the king knew that the time would come when his four sons inherit the kingdom. Though he loved his sons and was proud of their peacefulness and sense of justice, he was well aware of their rivalry when it came to power and feared that a struggle of his sons about his legacy might plunge his kingdom into civil war. To prevent this from happening, he decided to meet with his sons to arrange a subdivision of his kingdom. After a long and exhausting discussion, they had succeeded to divide the villages, towns, and castles into four partitions such that each son was satisfied with the places he was promised. At this night, the king fell asleep with a smile on his face, knowing that his kingdom would be cared for after his death.

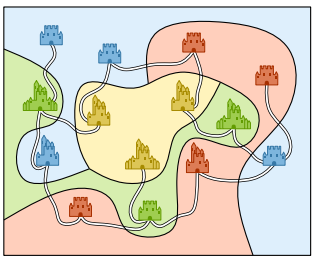


In the next morning the king awoke with the strange feeling that he had done a mistake. At the previous day he had decided which son should rule which place without dividing the surrounding terrain. Knowing that his sons did not care so much about the unsettled terrain, the king decided to do this subdivision without another discussion with his sons. He asked his cartographer for a map of his kingdom and colored the places in the map in the sons' colors.

After he finished, he immediately spotted his mistake: The places of each son were scattered over the whole kingdom, making it impossible for each son to transport goods between two places without passing through the place of a brother. The king feared that the sons would charge each other high fees for passing through their places, inevitably leading to conflicts.

After thinking for a moment, the king had the idea to subdivide his kingdom into four connected regions, each containing the places of one son. In this way, each son would be able to avoid excessive fees by building new roads directly connecting his places. The king actually hoped that the potential to build these roads would suffice to keep the fees at a reasonable level.

When trying to implement his idea, he encountered the problem that choosing a region for one son sometimes made it impossible to choose a connected region for another son. He had the feeling that it should be possible to choose a connected region for each son but he was not sure about it. To not waste too much time in finding something non-existing, he called for his mathematician, described his problem and asked, "Can I be sure to find a solution if I search for a while or may this problem be unsolvable?" The mathematician thought for a moment before he answered, "You can be sure to find a solution. The problem here is that one gets easily distracted by the geometry although your problem is of topological nature."



With this useful hint, it did not take long for the king to find a solution to his problem. At the end of the day, the king went to bed with the good feeling that he spotted and corrected his mistake from the previous day before it was too late.

At the next day, he decided to take another look at the map before showing it to his sons. He already convinced himself that he chose a good way of dividing his kingdom when he spotted an issue: The blue place in the east had a road to a green place passing through the red region. Thus, trading goods between these two places would require

to pass through the land of a son not involved in the trade. This son might then require a fee for passing through his land and the king did not like that. To make it even worse, the king spotted two more places where this issue occurred: A road connecting a green and a yellow place passing through the blue region in the west, and a road connecting a blue and a red place passing through the green region in the south-west.

Before trying to rearrange the regions, the king again called for his mathematician and asked him, "I'm not satisfied with just finding a connected region for each son's places. I also want no road to passes through a region. Is that possible?" The mathematician thought for a moment before he answered, "It looks like you encountered a difficult problem. I'm pretty sure that the places could have been divided into four, such that it is impossible to find such regions. Whether it is possible with the partitioning you and your sons agreed upon? I don't know yet. But I will think about it."

Will the king's mathematician find a solution to this problem? Will such a solution ensure a peaceful future for the kingdom? And most importantly, could a computer have helped the king to solve this problem? Find out in the conclusion of this chapter at page 164!

As most people do not have a Kingdom that needs to be separated, the more realistic motivation for considering the problem Clustered Planarity is the wish to visualize graphs whose nodes are hierarchically structured. In this settings, one usually has two objectives. First, the graph should be drawn nicely. Second, the hierarchical structure should be expressed by the drawing. Regarding the first objective, we require planar drawings as the number of crossings in a drawing of a graph is a major aesthetic criterion. A natural way to represent a cluster is a simple (= simply connected) region containing exactly the vertices in the cluster. To express the hierarchical structure, the boundaries of two regions must not cross and edges of the graph can cross region boundaries at most once, namely if only one of its endpoints lies inside the cluster. Recall that such a drawing is called *c-planar*; see the preliminaries in Section 1.4.6 for a formal definition. Determining the computational complexity of the problem Clustered Planarity of recognizing c-planar clustered graphs is a fundamental open question in the field of Graph Drawing.

Clustered Planarity was first considered by Lengauer [Len89] but in a completely different context. He gave an efficient algorithm for the case that every cluster is connected. Feng et al. [FCE95b], who coined the name c-planarity, rediscovered the problem and gave a similar algorithm. They also presented an algorithm for actually drawing c-planar clustered graphs [FCE95a]. Cornelsen and Wagner [CW06] showed that Clustered Planarity is equivalent to testing planarity when every cluster and every co-cluster is connected.

Relaxing the condition that every cluster must be connected makes Clustered Planarity surprisingly difficult. Efficient algorithms are known only for very restricted cases and many of these algorithms are very involved. One example is the efficient algorithm by Jelínek et al. [Jel+09b; Jel+09a] for the case that every cluster consists of at most two connected components while the planar embedding of the underlying graph is fixed. Another efficient algorithm by Jelínek et al. [Jel+09c] solves the case that every cluster has at most four outgoing edges.

A popular restriction is to require a *flat* hierarchy, i.e., every pair of clusters has empty intersection. For example, Di Battista and Frati [DF08] solve the case where the clustering is flat, the graph has a fixed planar embedding and the size of the faces is bounded by five. Sections 6.3.1 and 6.3.2 contain additional related work viewed from the new perspective we present in this chapter.

Contribution and Outline

We first present the cd-tree data structure (Section 6.2), which is similar to a data structure used by Lengauer [Len89]. We use the cd-tree to characterize c-planarity in terms of a combinatorial embedding problem. We believe that our definition of the cd-tree together with this characterization provides a very useful perspective on Clustered Planarity and significantly simplifies some previous results.

In Section 6.3 we define different constrained-planarity problems. We use the cd-tree to show in Section 6.3.1 that these problems are equivalent to different versions of Clustered Planarity of flat-clustered graphs. We also discuss which cases of the constrained embedding problems are solved by previous results on c-planarity of flat-clustered graphs. Based on these insights, we derive a generic algorithm for Clustered Planarity with different restrictions in Section 6.3.2 and discuss previous work in this context.

In Section 6.4, we show how the cd-tree characterization together with results on the problem Simultaneous PQ-Ordering (see Chapter 5) leads to efficient algorithms for the cases that (i) every cluster and every co-cluster consists of at most two connected components; or (ii) every cluster has at most five outgoing edges. The latter extends the result by Jelínek et al. [Jel+09c], where every cluster has at most four outgoing edges.

6.2 The CD-Tree

Let (G,T) be a clustered graph. We introduce the *cd-tree (cut- or cluster-decomposition-tree)* by enhancing each node of T with a multi-graph that represents the decomposition of G along its cuts corresponding to edges in T; see Figure 6.1a and b for an example. We note that Lengauer [Len89] uses a similar structure. Our notation is inspired by SPQR-trees.

Let μ be a node of T with neighbors $\mu_1, \ldots, \mu_k$ and incident edges $\varepsilon_i = \{\mu, \mu_i\}$ (for $i = 1, \ldots, k$). Removing μ separates T into k subtrees $T_1, \ldots, T_k$. Let $V_1, \ldots, V_k \subseteq V$ be the vertices of G represented by leaves in these subtrees. The *skeleton* $\text{skel}(\mu)$ of μ is the multi-graph obtained from G by contracting each subset V_i into a single vertex v_i (the resulting graph has multiple edges but we remove loops). These contracted vertices v_i are called *virtual vertices*. Note that skeletons of inner nodes of T contain only virtual vertices, while skeletons of leaves consist of one virtual and one non-virtual vertex. The node μ_i is the neighbor of μ *corresponding* to v_i and the virtual vertex in $\text{skel}(\mu_i)$ corresponding to μ is the *twin* of v_i, denoted by $\text{twin}(v_i)$. Note that $\text{twin}(\text{twin}(v_i)) = v_i$.

The edges incident to v_i are exactly the edges of G crossing the cut that corresponds to the tree edge ε_i. Thus, the same edges of G are incident to v_i and $\text{twin}(v_i)$. This gives a bound on the total size c of the cd-tree's skeletons (which we briefly call the *size of the cd-tree*). The total number of edges in skeletons of T is twice the total size of all cuts represented by T. As edges might cross a linear number of cuts (but obviously not more), the size of the cd-tree is at most quadratic in the number of vertices of G, i.e., $c \in O(n^2)$.

Assume the cd-tree is rooted. Recall that in this case every node μ represents a cluster of G. In analogy to the notion for SPQR-trees, we define the *pertinent graph* $\text{pert}(\mu)$ of the node μ to be the cluster represented by μ. Note that one could also define the pertinent graph recursively, by removing the virtual vertex corresponding to the parent of μ (the *parent vertex*) from $\text{skel}(\mu)$ and replacing each remaining virtual vertex by the pertinent graph of the corresponding child of μ. Clearly, the pertinent graph of a leaf of T is a single vertex and the pertinent graph of the root is the whole graph G. A similar concept, also defined for unrooted cd-trees, is the *expansion graph*. The expansion graph $\text{exp}(v_i)$ of a virtual vertex v_i in $\text{skel}(\mu)$ is the pertinent graph of its corresponding neighbor μ_i of μ, when rooting T at μ. One can think of the expansion graph $\text{exp}(v_i)$ as the subgraph of G represented by v_i in $\text{skel}(\mu)$. In the following we use the rooted and unrooted points of view interchangeably.

The leaves of a cd-tree represent singleton clusters that exist only due to technical reasons. It is often more convenient to consider cd-trees with all leaves removed as follows. Let μ be a node with virtual vertex v in $\text{skel}(\mu)$ that corresponds to a leaf. The leaf contains $\text{twin}(v)$ and a non-virtual vertex $v \in V$ in its skeleton (with an edge between $\text{twin}(v)$ and v for each edge incident to v in G). We replace v in $\text{skel}(\mu)$ with the non-virtual vertex v and remove the leaf containing v. Clearly, this preserves all clusters except for the singleton cluster. Moreover, the graph G represented by the cd-tree remains unchanged as we replaced the virtual vertex v by its expansion graph $\text{exp}(v) = v$. In the following we always assume the leaves of cd-trees to be removed.

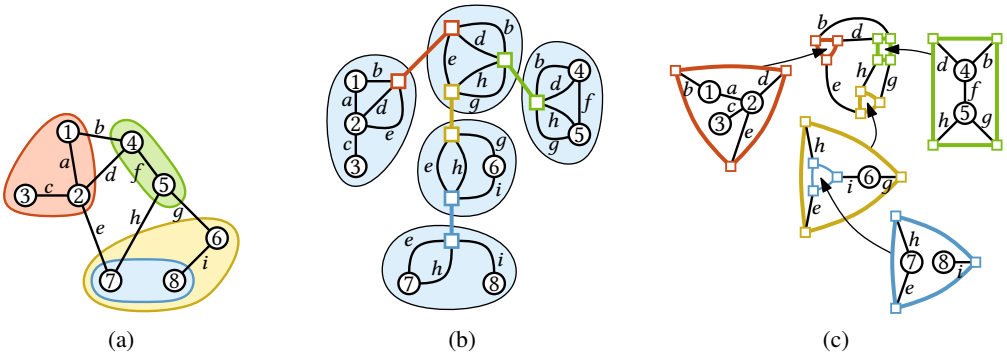


Figure 6.1: (a) A c-planar drawing of a clustered graph. (b) The corresponding (rooted) cd-tree (without leaves). The skeletons are drawn inside their corresponding (blue) nodes. Every pair of twins (boxes with the same color) has the same edge-ordering. (c) Construction of a c-planar drawing from the cd-tree.

The CD-Tree Characterization

We show that Clustered Planarity can be expressed in terms of edge-orderings in embeddings of the skeletons of T.

Theorem 6.1. *A clustered graph is c-planar if and only if the skeletons of all nodes in its cd-tree can be embedded such that every virtual vertex and its twin have the same edge-ordering.*

Proof. Assume G admits a c-planar drawing Γ on the sphere. Let μ be a node of T with incident edges $\varepsilon_1, \dots, \varepsilon_k$ connecting μ to its neighbors $\mu_1, \dots, \mu_k$, respectively. Let further v_i be the virtual vertex in $\text{skel}(\mu)$ corresponding to μ_i and let V_i be the nodes in the expansion graph $\exp(v_i)$. For every cut (V_i, V_i') (with $V_i' = V \setminus V_i$), Γ contains a simple closed curve C_i representing it. Since the V_i are disjoint, we can choose a point on the sphere to be the outside such that V_i lies inside C_i for $i = 1, \dots, k$. Since Γ is a c-planar drawing, the C_i do not intersect and only the edges of G crossing the cut (V_i, V_i') cross C_i exactly once. Thus, one can contract the inside of C_i to a single point while preserving the embedding of G. Doing this for each of the curves C_i yields the skeleton $\text{skel}(\mu)$ together with a planar embedding. Moreover, the edge-ordering of the vertex v_i is the same as the order in which the edges cross the curve C_i, when traversing C_i in clockwise direction. Applying the same construction for the neighbour μ_i corresponding to v_i yields a planar embedding of $\text{skel}(\mu_i)$ in which the edge-ordering of $\text{twin}(v_i)$ is the same as the order in which these edges cross the curve C_i, when traversing C_i in counter-clockwise direction. Thus, in the resulting embeddings of the skeletons, the edge-ordering of a virtual vertex and its twin is the same up to reversal. To make them the same one can choose a 2-coloring of T and mirror the embeddings of all skeletons of nodes in one color class.

Conversely, assume that all skeletons are embedded such that every virtual vertex and its twin have the same edge-ordering. Let μ be a node of T. Consider a virtual vertex v_i of skel(μ) with edge-ordering $e_1, \dots, e_\ell$. We replace v_i by a cycle $C_i = (v_i^1, \dots, v_i^\ell)$ and attach the edge e_j to the vertex v_i^j; see Figure 6.1c. Recall that twin(v_i) has in skel(μ_i) the same incident edges $e_1, \dots, e_\ell$ and they also appear in this order around twin(v_i). We also replace twin(v_i) by a cycle of length ℓ. We say that this cycle is the *twin* of C_i and denote it by twin(C_i) = (twin(v_i^1), …, twin(v_i^ℓ)) where twin(v_i^j) denotes the new vertex incident to the edge e_j. As the interiors of C_i and twin(C_i) are empty, we can glue the skeletons skel(μ) and skel(twin(μ)) together by identifying the vertices of C_i with the corresponding vertices in twin(C_i) (one of the mbeddings has to be flipped). Applying this replacement for every virtual vertex and gluing it with its twin leads to an embedded planar graph G^+ with the following properties. First, G^+ contains a subdivision of G. Second, for every cut corresponding to an edge $\varepsilon = \{\mu, \mu_i\}$ in T, G^+ contains the cycle C_i with exactly one subdivision vertex of an edge e of G if the cut corresponding to ε separates the endpoints of e. Third, no two of these cycles share a vertex. The planar drawing of G^+ gives a planar drawing of G. Moreover, the drawings of the cycles can be used as curves representing the cuts, yielding a c-planar drawing of G. □

Cutvertices in Skeletons

We show that cutvertices in skeletons correspond to different connected components in a cluster or in a co-cluster. More precisely, a cutvertex directly implies disconnectivity, while the opposite is not true. Consider the example in Figure 6.1. The cutvertex in the skeleton containing the vertices 7 and 8 corresponds to the two connected components in the blue cluster (containing 7 and 8). However, the two connected components in the orange cluster (containing 6–8) do not yield a new cutvertex in the skeleton containing the vertex 6. The following lemma in particular shows that requiring every cluster to be connected implies that the parent vertices of skeletons cannot be cutvertices.

Lemma 6.1. *Let v be a virtual vertex that is a cutvertex in its skeleton. The expansion graphs of virtual vertices in different blocks incident to v belong to different connected components in* exp(twin(v)).

Proof. Let μ be the node whose skeleton contains v. Recall that one can obtain the graph exp(twin(v)) by removing v from skel(μ) and replacing all other virtual vertices of skel(μ) with their expansion graphs. Clearly, this yields (at least) one different connected component for each of the blocks incident to v. □

Lemma 6.2. *Every cluster in a clustered graph is connected if and only if in every node μ of the rooted cd-tree the parent vertex is not a cutvertex in* skel(μ).

Proof. By Lemma 6.1, the existence of a cutvertex implies a disconnected cluster. Conversely, let $\text{pert}(\mu)$ be disconnected and assume without loss of generality that $\text{pert}(\mu_i)$ is connected for every child $\mu_1, \dots, \mu_k$ of μ in the cd-tree. One obtains $\text{skel}(\mu)$ without the parent vertex v by contracting in $\text{pert}(\mu)$ the child clusters $\text{pert}(\mu_i)$ to virtual vertices v_i. As the contracted graphs $\text{pert}(\mu_i)$ are connected while the initial graph $\text{pert}(\mu)$ is not, the resulting graph must be disconnected. Thus, v is a cutvertex in $\text{skel}(\mu)$. □

6.3 Clustered and Constrained Planarity

We first describe several constraints on planar embeddings, each restricting the edge-orderings of vertices. We then show the relation to Clustered Planarity.

Consider a finite set S (e.g., edges incident to a vertex). Denote the set of all cyclic orders of S by O_S. An *order-constraint* on S is simply a subset of O_S (only the orders in the subset are *allowed*). A *family of order-constraints* for the set S is a set of different order constraints, i.e., a subset of the power set of O_s. We say that a family of order-constraints has a *compact representation*, if one can specify every order-constraint in this family with polynomial space (in $|S|$). In the following we describe families of order-constraints with compact representations.

A *partition-constraint* is given by partitioning S into subsets $S_1 \cup \dots \cup S_k = S$. It requires that no two partitions *alternate*, i.e., elements $a_i, b_i \in S_i$ and $a_j, b_j \in S_j$ must not appear in the order a_i, a_j, b_i, b_j. A *PQ-constraint* requires that the order of elements in S is represented by a given PQ-tree with leaves S. A *full-constraint* contains only one order, i.e., the order of S is completely fixed.

A *partitioned full-constraint* restricts the orders of elements in S according to a partition constraint (partitions must not alternate) and additionally completely fixes the order within each partition. Similarly, *partitioned PQ-constraints* require the elements in each partition to be ordered according to a PQ-constraint. Clearly, this notion of partitioned order-constraints generalizes to arbitrary order-constraints.

Consider a planar graph G. By *constraining* a vertex v of G, we mean that there is an order-constraint on the edges incident to v. We then only allow planar embeddings of G where the edge-ordering of v is allowed by the order-constraint. By constraining G, we mean that several (or all) vertices of G are constrained.

6.3.1 Flat-Clustered Graph

Consider a flat-clustered graph, i.e., a clustered graph where the cd-tree is a star. We choose the center μ of the star to be the root. Let $v_1, \dots, v_k$ be the virtual vertices in $\text{skel}(\mu)$ corresponding to the children $\mu_1, \dots, \mu_k$ of μ. Note that $\text{skel}(\mu_i)$ contains exactly one virtual vertex, namely $\text{twin}(v_i)$. The possible ways to embed $\text{skel}(\mu_i)$ restrict the possible edge-orderings of $\text{twin}(v_i)$ and thus, by the characterization in

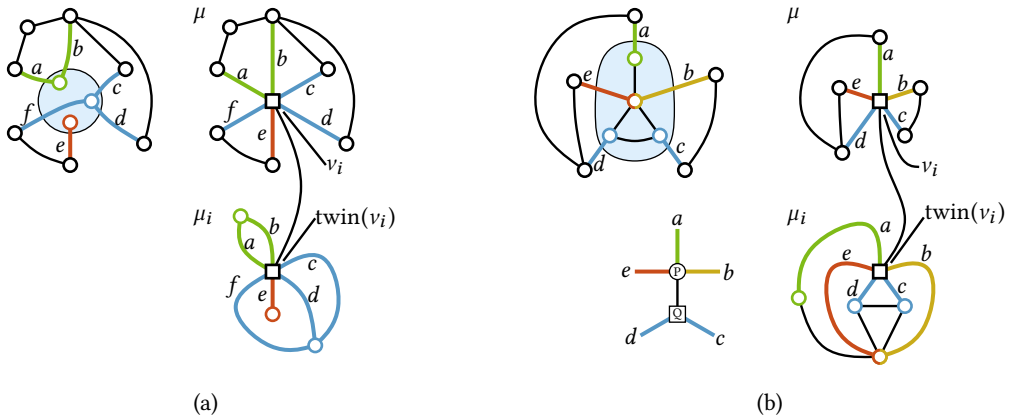


Figure 6.2: (a) A graph with a single cluster consisting of isolated vertices together with an illustration of its cd-tree. An edge-ordering of twin(v_i) corresponds to a planar embedding of skel(μ_i) if and only if no two partitions of the partitioning $\{\{a,b\},\{c,d,f\},\{e\}\}$ alternate. (b) A graph with a single connected cluster and its cd-tree. The valid edge-orderings of twin(v_i) are represented by the shown PQ-tree.

Theorem 6.1, the edge-orderings of v_i in skel(μ). Hence, the graph skel(μ_i) essentially yields an order constraint for v_i in skel(μ). We consider Clustered Planarity with differently restricted instances, leading to different families of order-constraints. To show that Clustered Planarity is equivalent to testing whether skel(μ) is planar with respect to order-constraints of a specific family, we have to show two directions. First, the embeddings of skel(μ_i) only yield order-constraints of the given family. Second, we can get every possible order-constraint of the given family by choosing an appropriate graph for skel(μ_i).

Theorem 6.2. *Clustered Planarity for flat-clustered graphs* (i) where each proper cluster consists of isolated vertices; (ii) where each cluster is connected; (iii) with fixed planar embedding; (iv) without restriction *is linear-time equivalent to testing planarity of a multi-graph with* (i) partition-constraints; (ii) PQ-constraints; (iii) partitioned full-constraints; (iv) partitioned PQ-constraints, *respectively.*

Proof. We start with case (i); see Figure 6.2a. Consider a flat-clustered graph G and let μ_i be one of the leaves of the cd-tree. As pert(μ_i) is a proper cluster, it consists of isolated vertices. Thus, skel(μ_i) is a set of vertices $v_1,\dots,v_\ell$, each connected (with multiple edges) to the virtual vertex twin(v_i). The vertices $v_1,\dots,v_\ell$ partition the edges incident to twin(v_i) into ℓ subsets. Clearly, in every planar embedding of skel(μ_i) no two partitions alternate. Moreover, every edge-ordering of twin(v_i) in which no two partitions alternate gives a planar embedding of skel(μ_i). Thus, the edges incident to v_i in skel(μ) are constrained by a partition-constraint, where the partitions are determined by the incidence of the edges to the vertices $v_1,\dots,v_\ell$. One can easily

construct the resulting instance of planarity with partition-constraints problem in linear time in the size of the cd-tree. Note that the cd-tree has linear size in G for flat-clustered graphs.

Conversely, given a planar graph H with partition-constraints, we set $\text{skel}(\mu) = H$. For every vertex of H we have a virtual vertex v_i in $\text{skel}(\mu)$ with corresponding child μ_i. We can simulate every partitioning of the edges incident to v_i by connecting edges incident to $\text{twin}(v_i)$ (in $\text{skel}(\mu_i)$) with vertices such that two edges are connected with the same vertex if and only if they belong to the same partition. Clearly, this construction runs in linear time.

Case (ii) is illustrated in Figure 6.2b. By Lemma 6.2 the condition of connected clusters is equivalent to requiring that the virtual vertex $\text{twin}(v_i)$ in the skeleton of any leaf μ_i of the cd-tree is not a cutvertex. The statement of the theorem follows from the fact that the possible edge-orderings of non-cutvertices is PQ-representable and that any PQ-tree can be achieved by choosing an appropriate planar graph in which $\text{twin}(v_i)$ is not a cutvertex (see Section 5.7).

In case (iii) the embedding of G is fixed. As in case (i), the blocks incident to $\text{twin}(v_i)$ in $\text{skel}(\mu_i)$ partition the edges incident to v_i in $\text{skel}(\mu)$ such that two partitions must not alternate. Moreover, the fixed embedding of G fixes the edge-ordering of non-virtual vertices and thus it fixes the embeddings of the blocks in $\text{skel}(\mu_i)$. Hence, we get partitioned full-constraints for v_i. Conversely, we can construct an arbitrary partitioned full-constraint as in case (i).

For case (iv) the arguments from case (iii) show that we again get partitioned order-constraints, while the arguments from case (ii) show that these order-constraints (for the blocks) are PQ-constraints. □

Related Work

Biedl [Bie98] proposes different drawing-models for graphs whose vertices are partitioned into two subsets. The model matching the requirements of c-planar drawings is called *HH-drawings*. Biedl et al. [BKM98] show that one can test for the existence of HH-drawings in linear time. Hong and Nagamochi [HN14] rediscovered this result in the context of 2-page book embeddings. These results solve Clustered Planarity for flat-clustered graphs if the skeleton of the root node contains only two virtual vertices. This is equivalent to testing planarity with partitioned PQ-constraints for multi-graphs with only two vertices (Theorem 6.2). Thus, to solve Clustered Planarity for flat-clustered graphs, one needs to solve an embedding problem on general planar multi-graphs that is so far only solved on a set of parallel edges (with absolutely non-trivial algorithms). This indicates that we are still far away from solving Clustered Planarity even for flat-clustered graphs.

Cortese et al. [Cor+05] give a linear-time algorithm solving Clustered Planarity for a flat-clustered cycle (i.e., G is a simple cycle) if the skeleton of the cd-tree's root

is a multi-cycle. The requirement that G is a cycle implies that the skeleton of each non-root node in T has the property that the blocks incident to the parent vertex are simple cycles. Thus, in terms of constrained planarity, they show how to test planarity of multi-cycles with partition-constraints where each partition has size two. The result can be extended to a special case of Clustered Planarity where the clustering is not flat. However, the cd-tree fails to have easy-to-state properties in this case, which shows that the cd-tree perspective of course has some limitations. Later, Cortese et al. [Cor+09b] extended this result to the case where G is still a cycle, while the skeleton of the root can be an arbitrary planar multi-graph that has a fixed embedding up to the ordering of parallel edges. This is equivalent to testing planarity of such a graph with partition-constraints where each partition has size two.

Jelínková et al. [Jel+09d] consider the case where each cluster contains at most three vertices (with additional restrictions). Consider a cluster containing only two vertices u and v. If u and v are connected, then the region representing the cluster can be always added and we can omit the cluster. Otherwise, the region representing the cluster in a c-planar drawing implies that one can add the edge uv to G, yielding an equivalent instance. Thus, one can assume that every cluster has size exactly 3, which yields flat-clustered graphs. In this setting they give efficient algorithms for the cases that G is a cycle and G is 3-connected. Moreover, they give an FPT-algorithm for the case that G is an *Eulerian graph* with k nodes, i.e., a graph obtained from a 3-connected graph of size k by multiplying and then subdividing edges.

In case G is 3-connected, its planar embedding is fixed and thus the edge-ordering of non-virtual vertices is fixed. Thus, one obtains partitioned full-constraints with the restriction that there are only three partitions. Clearly, the requirement that G is 3-connected also restricts the possible skeletons of the root of the cd-tree. It is an interesting open question whether planarity with partitioned full-constraints with at most three partitions can be tested efficiently for arbitrary planar graphs. In case G is a cycle, one obtains partition constraints with only three partitions and each partition has size two. Note that this in particular restricts the skeleton of the root to have maximum degree 6. Although these kind of constraints seem pretty simple to handle, the algorithm by Jelínková et al. is pretty involved. It seems like one barrier where constrained embedding becomes difficult is when there are partition constraints with three or more partitions (see also Theorem 6.4). The result about Eulerian graphs in a sense combines the cases where G is 3-connected and a cycle. A vertex has either degree two and thus yields a partition of size two or it is one of the constantly many vertices with higher degree for which the edge-ordering is partly fixed.

Chimani et al. [Chi+14] give a polynomial time algorithm for embedded flat-clustered graphs with the additional requirement that each face is incident to at most two vertices of the same cluster. This basically solves planarity with partitioned full-constraints with some additional requirements. We do not describe how these additional requirements

exactly restrict the possible instances of constrained planarity. However, we give some properties that shed a light on why these requirements make planarity with partitioned full-constraints tractable.

Consider the skeleton skel(μ) of a (non-root) node μ of the cd-tree. As G is a flat-clustered graph, skel(μ) has only a single virtual vertex. Assume we choose planar embeddings with consistent edge orderings for all skeletons (i.e., we have a c-planar embedding of G). Two non-virtual vertices u and v in skel(μ) that are incident to the same face of skel(μ) are then also incident to the same face of G. Note that the converse is not true, as a vertex adjacent to the virtual vertex of skel(μ) can be incident to the same face as any other vertex with this property. As the non-virtual vertices of skel(μ) belong to the same cluster, at most two of them can be incident to a common face of skel(μ). Thus, every face of skel(μ) has at most two non-virtual vertices on its boundary. One implication of this fact is that every connected component of the cluster is a tree for the following reason. If a connected component contains a cycle, it has at least two faces with more than two vertices on the boundary. In skel(μ) only one of the two faces can be split into several faces by the virtual vertex, but the other face remains.

More importantly, the possible ways how the blocks incident to the virtual vertex of skel(μ) can be nested into each other is heavily restricted. In particular, embedding multiple blocks next to each other into the same face of another block is not possible, as this would result in a face of skel(μ) with more than two non-virtual vertices on its boundary. In a sense, this enforces a strong nesting of the blocks. Thus, one actually obtains a variant of planarity with partitioned full-constraints, where the way how the partitions can nest is restricted beyond forbidding two partitions to alternate. These and similar restrictions on how partitions are allowed to be nested lead to a variety of new constrained planarity problems. We believe that studying those restricted problems can help to deepen the understanding of the more general partitioned full-constraints or even partitioned PQ-constraints.

6.3.2 General Clustered Graphs

Expressing c-planarity for general clustered graphs (not necessarily flat) in terms of constrained planarity problems is harder for the following reason. Consider a leaf μ in the cd-tree. The skeleton of μ is a planar graph yielding (as in the flat-clustered case) partitioned PQ-constraints for its parent μ'. This restricts the possible embeddings of skel(μ') and thus the order-constraints one obtains for the parent of μ' are not necessarily again partitioned PQ-constraints.

One can express this issue in the following, more formal, way. Let G be a planar multi-graph with vertices $v_1, \dots, v_n$ and designated vertex $v = v_n$. The map φ_G^v maps a tuple $(C_1, \dots, C_n)$ where C_i is an order-constraint on the edges incident to v_i to an order-constraint C on the edges incident to v. The order-constraint $C = \varphi_G^v(C_1, \dots, C_n)$

contains exactly those edge-orderings of v that one can get in a planar embedding of G that respects $C_1, \ldots, C_n$. Note that C is empty if and only if there is no such embedding. Note further that testing planarity with order-constraints is equivalent to deciding whether φ_G^v evaluates to the empty set. We call such a map φ_G^v a *constrained-embedding operation*.

The issue mentioned above (with constraints iteratively handed to the parents) boils down to the fact that partitioned PQ-constraints are not closed under constrained-embedding operations. On the positive side, we obtain a general algorithm for solving Clustered Planarity as follows. Assume we have a family of order-constraints $\mathcal{C}$ with compact representations that is closed under constrained-embedding operations. Assume further that we can evaluate the constrained embedding operations in polynomial time on order-constraints in $\mathcal{C}$. Then one can simply solve Clustered Planarity by traversing the cd-tree bottom-up, evaluating for a node μ with parent vertex v the constrained-embedding operation $\varphi^v_{\text{skel}(\mu)}$ on the constraints one computed in the same way for the children of μ.

Clearly, when restricting the skeletons of the cd-tree or requiring properties for the parent vertices in these skeletons, these restrictions carry over to the constrained-embedding operations one has to consider. More precisely, let $\mathcal{R}$ be a set of pairs (G, v), where v is a vertex in G. We say that a clustered graph is *$\mathcal{R}$-restricted* if $(\text{skel}(\mu), v) \in \mathcal{R}$ holds for every node μ in the cd-tree with parent vertex v. Moreover, the *$\mathcal{R}$-restricted* constrained-embedding operations are those operations φ_G^v with $(G, v) \in \mathcal{R}$. The following theorem directly follows.

Theorem 6.3. *One can solve Clustered Planarity for $\mathcal{R}$-restricted clustered graphs in polynomial time if there is a family $\mathcal{C}$ of order-constraints such that*

- *$\mathcal{C}$ has a compact representation,*
- *$\mathcal{C}$ is closed under $\mathcal{R}$-restricted constrained-embedding operations,*
- *every $\mathcal{R}$-restricted constrained-embedding operation on order-constraints in $\mathcal{C}$ can be evaluated in polynomial time.*

When dropping the requirement that $\mathcal{C}$ has a compact representation the algorithm becomes super-polynomial only in the maximum degree d of the virtual vertices (the number of possible order-constraints for a set of size d depends only on d). Moreover, if φ_G^v has only k order constraints (whose sizes are bounded by a function of d) as input, then φ_G^v can be evaluated by iterating over all combinations of orders, applying a planarity test in every step. This gives an FPT-algorithm with parameter $d + k$ (running time $O(f(d + k)p(n))$, where f is a computable function depending only on $d + k$ and p is a polynomial). In other words, we obtain an FPT-algorithm where the parameter is the sum of the maximum degree of the tree T and the maximum number

of edges leaving a cluster. Note that this generalizes the FPT-algorithm by Chimani and Klein [CK13] with respect to the total number of edges connecting different clusters.

Moreover, Theorem 6.3 has the following simple implication. Consider a clustered graph where each cluster is connected. This restricts the skeletons of the cd-tree such that non of the parent vertices is a cutvertex (Lemma 6.1). Thus, we have $\mathcal{R}$-restricted clustered graphs where $(G, v) \in \mathcal{R}$ implies that v is not a cutvertex in G. PQ-constraints are closed under $\mathcal{R}$-restricted constrained-embedding operations as the valid edge-ordering of non-cutvertices is PQ-representable and planarity with PQ-constraints is basically equivalent to planarity (one can model a PQ-tree with a simple gadget). Thus, Theorem 6.3 directly implies that Clustered Planarity can be solved in polynomial time if each cluster is connected.

Related Work

The above algorithm resulting from Theorem 6.3 is more or less the one described by Lengauer [Len89]. The algorithm was later rediscovered by Feng et al. [FCE95b] who coined the term "c-planarity". The algorithm runs in $O(c) \subseteq O(n^2)$ time (recall that c is the size of the cd-tree). Dahlhaus [Dah98] improves the running time to $O(n)$. Cortese et al.[Cor+08] give a characterization that also leads to a linear-time algorithm.

Goodrich et al. [GLS06] consider the case where each cluster is either connected or *extrovert*. Let μ be a node in the cd-tree with parent μ'. The cluster pert(μ) is extrovert if the parent cluster pert(μ') is connected and every connected component in pert(μ) is connected to a vertex not in the parent pert(μ'). They show that one obtains an equivalent instance by replacing the extrovert cluster pert(μ) with one cluster for each of its connected components while requiring additional PQ-constraints for the parent vertex in the resulting skeleton. In this instance every cluster is connected and the additional PQ-constraints clearly do no harm.

Another extension to the case where every cluster must be connected is given by Gutwenger et al. [Gut+02]. They give an algorithm for the case where every cluster is connected with the following exception. Either, the disconnected clusters form a path in the tree or for every disconnected cluster the parent and all siblings are connected. This has basically the effect that at most one order-constraint in the input of a constrained-embedding operation is not a PQ-tree.

Jelínek et al. [Jel+09a; Jel+09b] assume each cluster to have at most two connected components and the underlying (connected) graph to have a fixed planar embedding. Thus, they consider $\mathcal{R}$-restricted clustered graphs where $(G, v) \in \mathcal{R}$ implies that v is incident to at most two different blocks. The fixed embedding of the graph yields additional restrictions that are not so easy to state within this model.

6.4 Cutvertices with Two Non-Trivial Blocks

The input of the Simultaneous PQ-Ordering problem consists of several PQ-trees together with child-parent relations between them (the PQ-trees are the nodes of a directed acyclic graph) such that the leaves of every child form a subset of the leaves of its parents. Simultaneous PQ-Ordering asks whether one can choose orders for all PQ-trees *simultaneously* in the sense that every child-parent relation implies that the order of the leaves of the parent are an extension of the order of the leaves of the child. In this way one can represent orders that cannot be represented by a single PQ-tree. For example, adding one or more children to a PQ-tree T restricts the set of orders represented by T by requiring the orders of different subsets of leaves to be represented by some other PQ-tree. Moreover, one can synchronize the orders of different trees that share a subset of leaves by introducing a common child containing these leaves.

Simultaneous PQ-Ordering is NP-hard but efficiently solvable for so-called 2-fixed instances [BR13]; see also Chapter 5. For every biconnected planar graph G, there exists an instance of Simultaneous PQ-Ordering, the *PQ-embedding representation*, that represents all planar embeddings of G; see Chapter 5. It has the following properties.

- For every vertex v in G there is a PQ-tree $T(v)$, the *embedding tree*, that has the edges incident to v as leaves.
- For every solution of the PQ-embedding representation, setting the edge-ordering of every vertex v to the order given by $T(v)$ yields a planar embedding. Moreover, one can obtain every embedding of G in this way.
- The instance remains 2-fixed when adding up to one child to each embedding tree.

A PQ-embedding representation still exists if every cutvertex in G is incident to at most two *non-trivial blocks* (blocks that are not just bridges) [BR11].

Theorem 6.4. *Clustered Planarity can be solved in $O(c^2) \subseteq O(n^4)$ time if every virtual vertex in the skeletons of the cd-tree is incident to at most two non-trivial blocks.*

Proof. Let G be a clustered graph with cd-tree T. For the skeleton of each node in T, we get a PQ-embedding representation with the above-mentioned properties. Let μ be a node of T and let v be a virtual vertex in $\text{skel}(\mu)$. By the above properties, the embedding representation of μ contains the embedding tree $T(v)$ representing the valid edge-orderings of v. Moreover, for $\text{twin}(v)$ there is an embedding tree $T(\text{twin}(v))$ in the embedding representation of the skeleton containing $\text{twin}(v)$. To ensure that v and $\text{twin}(v)$ have the same edge-ordering, one can simply add a PQ-tree as common child of $T(v)$ and $T(\text{twin}(v))$. We do this for every virtual node in the skeletons of T.

Due to the last property of the PQ-embedding representations, the resulting instance remains 2-fixed and can thus be solved efficiently.

Every solution of this Simultaneous PQ-Ordering instance D yields planar embeddings of the skeletons such that every virtual vertex and its twin have the same edge-ordering. Conversely, every such set of embeddings yields a solution for D. It thus follows by the characterization in Theorem 6.1 that solving Clustered Planarity is equivalent to solving D. The size of D is linear in the size c of the cd-tree T. Moreover, solving Simultaneous PQ-Ordering for 2-fixed instances can be done in quadratic time [BR13], yielding the running time $O(c^2)$. □

Theorem 6.4 includes the following interesting cases. The latter extends the result by Jelínek et al. [Jel+09c] from four to five outgoing edges per cluster.

Corollary 6.1. *Clustered Planarity can be solved in $O(c^2) \subseteq O(n^4)$ time if every cluster and every co-cluster has at most two connected components.*

Proof. Note that the expansion graphs of nodes in skeletons of T are exactly the clusters and co-clusters. Thus, the expansion graphs consist of at most two connected components. By Lemma 6.1 the cutvertices in skeletons of T are incident to at most two different blocks. Thus, we can simply apply Theorem 6.4. □

Corollary 6.2. *Clustered Planarity can be solved in $O(n^2)$ time if every cluster has at most five outgoing edges.*

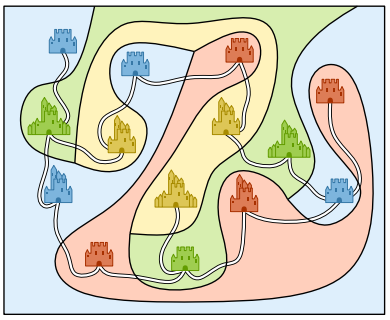
Proof. Let μ be a node with virtual vertex v in its skeleton. The edges incident to v in $\text{skel}(\mu)$ are exactly the edges that separate $\exp(v)$ from the rest of the graph $\exp(\text{twin}(v))$. Thus, if every cluster has at most five outgoing edges, the virtual vertices in skeletons of T have maximum degree 5. With five edges incident to a vertex v, one cannot get more than two non-trivial blocks incident to v. It follows from Theorem 6.4 that we can solve Clustered Planarity in $O(c^2)$ time. As we have a linear number of cuts, each of constant size (at most 5), we get $c \in O(n)$. □

6.5 Conclusion

In this chapter we introduced the cd-tree and showed that it can be used to reformulate the classic problem Clustered Planarity as a constrained embedding problem. Afterwards, we interpreted several previous results on Clustered Planarity from this new perspective. In many cases the new perspective simplifies these algorithms or at least gives a better intuition why the imposed restrictions are helpful towards making the problem tractable. In some cases the new view allowed us to generalize and extend previous results to larger sets of instances.

We believe that the constrained embedding problems we defined provide a promising starting point for further research, e.g., by studying restricted variants to further deepen the understanding of the problem Clustered Planarity.

It remains to answer the open questions from the story about the king and his sons. Did the king's mathematician find a solution? Indeed, he did, and it probably helped to ensure a peaceful coexistence of the four sons. Would a computer have helped to find this solution? As this particular instance is small, a brute-force algorithm can solve it in reasonable time: When considering the cd-tree, the skeleton of the root cluster is K_4 with multiple edges. Thus, the embedding choices consist of reordering parallel edges. As there are four edges between the blue and red cluster, three edges between the blue and the green, three between the green and the yellow, and two between the green and the red cluster, there are only $4! \cdot 3! \cdot 3! \cdot 2! = 1728$ embeddings of the root cluster. However, a brute-force approach is no longer feasible for kings with a larger kingdom. Unfortunately, we do not know whether there exists an efficient algorithm for the king's problem. In fact, the king needs to solve Clustered Planarity for a flat clustered planar graph with fixed planar embedding, which is equivalent to planarity with partitioned full-constraints.



7 Disconnectivity in Simultaneous Planarity

Previous approaches to the problem of testing whether two graphs $G^{①}$ and $G^{②}$ sharing a common graph G are simultaneously planar always assume the common graph to be connected. This restriction makes it sufficient to find planar embeddings of $G^{①}$ and $G^{②}$ that imply the same edge orderings in G. If G has multiple connected components, one additionally has to ensure that the relative positions of these components with respect to each other are consistent. In this chapter, we consider the case that $G^{①}$, $G^{②}$, and G are not necessarily connected. In particular, we provide techniques for ensuring consistent relative positions.

First, we show that a general instance of the problem Sefe (which asks whether $G^{①}$ and $G^{②}$ are simultaneously planar; recall the definitions from the preliminaries in Section 1.4.7) can be reduced in linear time to an equivalent instance where $G^{①}$ and $G^{②}$ have the same vertex set and are both connected. Second, for the case where G is the disjoint union of cycles, we introduce the *CC-tree* which represents all embeddings of G that extend to planar embeddings of $G^{①}$. We show that CC-trees can be computed in optimal linear time, and that their intersection is again a CC-tree. This yields a linear-time algorithm for Sefe if G consists of cycles. Moreover, this algorithm directly extends to the sunflower case where multiple graphs intersect in the same common graph G. These results, including the CC-tree, extend to the case where G consists of arbitrary connected components, each with a fixed planar embedding on the sphere. Then the running time is $O(n^2)$.

This chapter is based on joint work with Ignaz Rutter [BR15].

7.1 Introduction

The problem Sefe and its variants, such as Simultaneous Geometric Embedding, where one insists on a simultaneous straight-line drawing, have been studied intensively in the past years; see our recent survey [BKR13b] for an overview. As there are planar graphs that cannot be embedded simultaneously although each of them is planar, the question of deciding whether given graphs admit a simultaneous embedding is of high interest. Gassner et al. [Gas+06] show that it is NP-complete to decide Sefe for three or more graphs. For two graphs the complexity status is still open.

However, there are several approaches yielding efficient algorithms for special cases. Fowler at al. [Fow+09] show how to solve Sefe efficiently if $G^{①}$ and G have at most two and one cycle, respectively. Fowler et al. [Fow+11] characterize the class of common

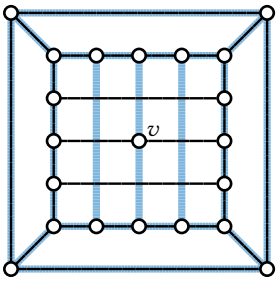


Figure 7.1: An instance of Sefe that admits a simultaneous embedding. This is no longer true if the vertex v, which is an isolated vertex in the common graph, is connected to the rest of the common graph by adding edges. The graph $G^{①}$ is thin and black; $G^{②}$ is bold and blue; the common graph is both.

graphs that always admit a simultaneous embedding. Angelini et al. [Ang+10] show that if one of the input graphs has a fixed planar embedding, then Sefe can be solved in linear time. Haeupler et al. [HJL10] solve Sefe in linear time for the case that the common graph is biconnected. Angelini et al. [Ang+12] obtain the same result with a completely different approach. They additionally solve the case where the common graph is a star and, moreover, show the equivalence of the case where the common graph is connected to the case where the common graph is a tree and relate it to a constrained book embedding problem. Besides the results in Chapter 8, the currently least restrictive result (in terms of connectivity) is the algorithm by Bläsius and Rutter [BR13] that solves Sefe in polynomial time for the case that both graphs are biconnected and the common graph is connected.

These algorithms solving Sefe have in common that they use the result by Jünger and Schulz [JS09] stating that the question of finding a simultaneous embedding for two graphs is equivalent to the problem of finding planar embeddings of $G^{①}$ and $G^{②}$ such that they induce the same embedding on G. Moreover, they have in common that they all assume that the common graph is connected, implying that it is sufficient to enforce consistent edge orderings. Especially in the result by Bläsius and Rutter [BR13] this is heavily used, as they explicitly consider only orders of edges around vertices using PQ-trees; also see Chapter 5 about Simultaneous PQ-Ordering.

If the common graph is not connected, we additionally have to care about the relative positions of connected components to one another, which introduces an additional difficulty. Note that the case where the common graph is disconnected cannot be reduced to the case where it is connected by inserting additional edges; see Figure 7.1.

Approaches not relying on the characterization by Jünger and Schulz [JS09] have only appeared recently. Schaefer [Sch13] characterizes, for certain classes of Sefe instances, the pairs of graphs that admit a simultaneous embedding via the independent odd crossing number. Among others, this gives a polynomial-time algorithm for Sefe when the common graph has maximum degree 3 and is not necessarily connected.

Contribution

In this work we tackle Sefe from the opposite direction than the so far known results. We assume that the cyclic order of edges around vertices in G is already fixed and we only have to ensure that the embeddings chosen for the input graphs imply compatible relative positions for the common graph. Initially, we assume that the graph G consists of a set of disjoint cycles, each of them having a unique planar embedding. We present a novel data structure, the *CC-tree*, which represents all embeddings of a set of disjoint cycles that can be induced by an embedding of a graph containing them as a subgraph. We moreover show that two such CC-trees can be intersected, again yielding a CC-tree. Thus, for the case that $G^{①}$ and $G^{②}$ have the common graph G consisting of a set of disjoint cycles, the intersection of the CC-trees corresponding to $G^{①}$ and $G^{②}$ represents all simultaneous embeddings. We show that CC-trees can be computed and intersected in linear time, yielding a linear-time algorithm to solve Sefe for the case that the common graph consists of disjoint cycles. Note that this obviously also yields a linear-time algorithm to solve Sefe for more than two graphs if they all share the same common graph consisting of a set of disjoint cycles, i.e., if we have the sunflower case. We show that these results can be further extended to the case where the common graph may contain arbitrary connected components, each of them with a prescribed planar embedding. However, in this case the corresponding data structure, called CC$^{\oplus}$-tree, may have quadratic size. These results show that the choice of relative positions of several connected components does not solely make the problem Sefe hard to solve.

Note that these results have an interesting application concerning the problem Partially Embedded Planarity. The input of Partially Embedded Planarity is a planar graph G together with a fixed embedding for a subgraph H (including fixed relative positions). It asks whether G admits a planar embedding extending the embedding of H. Angelini et al. [Ang+10] introduced this problem and solve it in linear time. The CC$^{\oplus}$-tree can be used to solve Partially Embedded Planarity in quadratic time as it represents all possible relative positions of the connected components in H to one another that can be induced by an embedding of G. It is then easy to test whether the prespecified relative positions can be achieved. In fact, this solves the slightly more general case of Partially Embedded Planarity where not all relative positions have to be fixed.

The above described results have one restriction that was not mentioned so far. The graphs $G^{①}$ and $G^{②}$ are assumed to be connected, otherwise the approach we present does not work. Fortunately, we can show that both graphs of an instance of Sefe can always be assumed to be connected, even if all vertices are assumed to be common vertices (forming isolated vertices when not connected via a common edge). This shows that Sefe can be solved efficiently if the common graph consists of disjoint cycles without further restrictions on the connectivity. Moreover, it is an interesting

result on its own as it applies to arbitrary instances of Sefe, not only to the special case we primarily consider here.

Outline

In Section 7.2 we show that, for any given instance of Sefe, there exists an equivalent instance such that both input graphs are connected, even if each vertex is assumed to be a common vertex. With this result instances of Sefe can always be assumed to have this property. In Section 7.3 we show how to solve Sefe in linear time if the common graph consists of disjoint cycles, including a compact representation of all simultaneous embeddings. In Section 7.4 we show how to extend these results to solve Sefe in quadratic time for the case that the common graph consists of arbitrary connected components, each with a fixed planar embedding.

7.2 Connecting Disconnected Graphs

Let $G^{①} = (V, E^{①})$ and $G^{②} = (V, E^{②})$ be two planar graphs with common graph $G = (V, E)$ with $E = E^{①} \cap E^{②}$. We show that the problem Sefe can be reduced to the case where $G^{①}$ and $G^{②}$ are required to be connected. First note that the connected components of the union of $G^{①}$ and $G^{②}$ can be handled independently. Thus we can assume that $G^{①} \cup G^{②}$ is connected. We first ensure that $G^{①}$ is connected without increasing the number of connected components in $G^{②}$. Afterwards we can apply the same steps to $G^{②}$ to make it connected, maintaining the connectivity of $G^{①}$.

Assume $G^{①}$ and $G^{②}$ consist of $k^{①}$ and $k^{②}$ connected components, respectively. Since the union of $G^{①}$ and $G^{②}$ is connected, we can always find an edge $e^{②} = \{v_1, v_2\} \in E^{②}$ such that the vertices v_1 and v_2 belong to different connected components $H_1^{①}$ and $H_2^{①}$ in $G^{①}$. We construct the *augmented instance* $(G_+^{①}, G_+^{②})$ of Sefe with respect to the edge $e^{②}$ by introducing a new vertex v_{12} and new edges $e = \{v_1, v_{12}\} \in E$ and $e^{①} = \{v_{12}, v_2\} \in E^{①}$. Note that $G_+^{①}$ has $k^{①} - 1$ connected components since $H_1^{①}$ and $H_2^{②}$ are now connected via the two edges e and $e^{①}$. Moreover, the number $k^{②}$ of connected components in $G^{②}$ does not change, since the edge e connects the new vertex v_{12} to one of its connected components. It remains to show that the original instance and the augmented instance are equivalent.

Lemma 7.1. *Let $(G^{①}, G^{②})$ be an instance of Sefe and let $(G_+^{①}, G_+^{②})$ be the augmented instance with respect to the edge $e^{②} = \{v_1, v_2\}$. Then $(G^{①}, G^{②})$ and $(G_+^{①}, G_+^{②})$ are equivalent.*

Proof. If the augmented instance admits a simultaneous embedding, then obviously the original instance does. To show the other direction assume the original instance $(G^{①}, G^{②})$ has a simultaneous embedding $(\mathcal{E}^{①}, \mathcal{E}^{②})$ inducing the embedding $\mathcal{E}$ for the

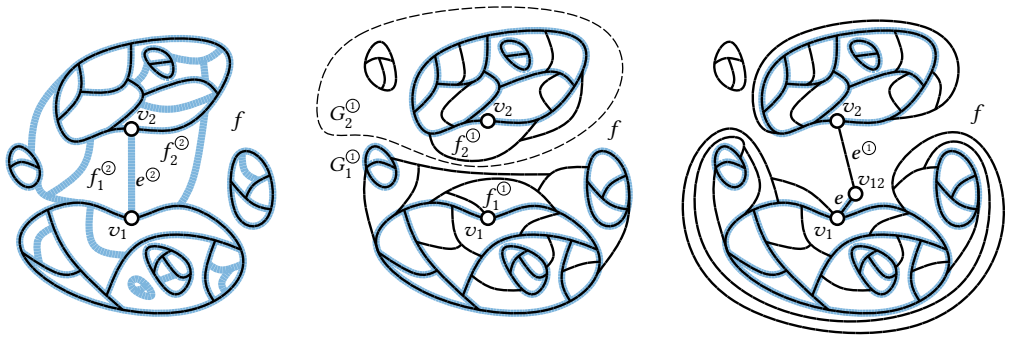


Figure 7.2: Illustration of Lemma 7.1; the common graph blue and black, $G^{①}$ is black and $G^{②}$ is blue. The graph $G^{②}$ with the edge $e^{②} = \{v_1, v_2\}$ lying in the common face f, which is the outer face of G (left). The graph $G^{①}$ with the faces $f_1^{①}, f_2^{①} \in \mathcal{F}^{①}(f)$ incident to v_1 and v_2, respectively, partitioned into $G_1^{①}$ and $G_2^{①}$ (middle). The resulting graph $G^{①}$ after choosing $f_i^{①}$ as outer face of $G_i^{①}$ (for $i = 1, 2$) and inserting the vertex v_{12} and the edges e and $e^{①}$ (right).

common graph. We show how to construct an embedding $\mathcal{E}'^{①}$ such that (i) $(\mathcal{E}'^{①}, \mathcal{E}^{②})$ is a simultaneous embedding, and (ii) the vertices v_1 and v_2 lie on the border of a common face in $\mathcal{E}'^{①}$. Then we can easily add the vertex v_{12} together with the two edges e and $e^{①}$, yielding a simultaneous embedding of the augmented instance $(G_+^{①}, G_+^{②})$. Note that the first property, namely that $(\mathcal{E}'^{①}, \mathcal{E}^{②})$ is a simultaneous embedding, is satisfied if and only if the embeddings $\mathcal{E}^{①}$ and $\mathcal{E}'^{①}$ induce the same embedding $\mathcal{E}$ on the common graph. Figure 7.2 illustrates the proof.

Consider a face f of the embedding $\mathcal{E}$ of the common graph. The embedding $\mathcal{E}^{①}$ of the graph $G^{①}$ splits this face f into a set of faces $\mathcal{F}^{①}(f) = \{f_1^{①}, \ldots, f_k^{①}\}$. We say that a face $f^{①} \in \mathcal{F}^{①}(f)$ is *contained* in f. Note that every face of $\mathcal{E}^{①}$ is contained in exactly one face of $\mathcal{E}$. The same definition can be made for the second graph.

The edge $e^{②} = \{v_1, v_2\}$ borders two faces $f_1^{②}$ and $f_2^{②}$ of $\mathcal{E}^{②}$. Since $e^{②}$ belongs exclusively to $G^{②}$ (otherwise v_1 and v_2 would not have been in different connected components in $G^{①}$) both faces $f_1^{②}$ and $f_2^{②}$ are contained in the same face f of the embedding $\mathcal{E}$ of the common graph G. We assume without loss of generality that f is the outer face. The face f may be subdivided by edges belonging exclusively to the graph $G^{①}$. However, we can find faces $f_1^{①}$ and $f_2^{①}$ of $\mathcal{E}^{①}$, both contained in f, such that v_1 and v_2 are contained in the boundary of these faces. If $f_1^{①} = f_2^{①}$ we are done since v_1 and v_2 lie on the boundary of the same face in $\mathcal{E}^{①}$. Otherwise, we split $G^{①}$ into two subgraphs $G_1^{①}$ and $G_2^{①}$ with the embeddings $\mathcal{E}_1^{①}$ and $\mathcal{E}_2^{①}$ induced by $\mathcal{E}^{①}$ as follows. The connected component $H_i^{①}$ (for $i = 1, 2$) containing v_i belongs to $G_i^{①}$ and all connected components that are completely contained in an internal face of $H_i^{①}$ also belong to $G_i^{①}$. All remaining connected components belong either to $G_1^{①}$ or to $G_2^{①}$. Note that this partition ensures that there is a simple closed curve in the outer face

of $\mathcal{E}^{①}$ separating $G_1^{①}$ and $G_2^{①}$. Thus, we can change the embeddings of $\mathcal{E}_1^{①}$ and $\mathcal{E}_2^{①}$ independently. In particular, we choose the faces $f_1^{①}$ and $f_2^{①}$ to be the new outer faces, yielding the changed embeddings $\mathcal{E}_1'^{①}$ and $\mathcal{E}_2'^{①}$, respectively. When combining these to embeddings by putting $G_1^{①}$ into the outer face of $G_2^{①}$ and vice versa, we obtain a new embedding $\mathcal{E}'^{①}$ of $G^{①}$ with the following two properties. First, the embedding induced for the common graph does not change since both faces $f_1^{①}$ and $f_2^{①}$ belong to the outer face f of the embedding $\mathcal{E}$ of the common graph G. Second, the vertices v_1 and v_2 both lie on the outer face of the embedding $\mathcal{E}'^{①}$. Hence, $(\mathcal{E}'^{①}, \mathcal{E}^{②})$ is still a simultaneous embedding of the instance $(G^{①}, G^{②})$ and the vertex v_{12} together with the two edges e and $e^{①}$ can be added easily, which concludes the proof. □

With this construction we can reduce the number of connected components of $G^{①}$ and $G^{②}$ and thus finally obtain an equivalent instance of Sefe in which both graphs are connected. We obtain the following Theorem.

Theorem 7.1. *For every instance $(G^{①}, G^{②})$ of Sefe there exits an equivalent instance $(G_{++}^{①}, G_{++}^{②})$ such that $G_{++}^{①}$ and $G_{++}^{②}$ are connected. Such an instance can be computed in linear time.*

Proof. Lemma 7.1 directly implies that an equivalent instance $(G_{++}^{①}, G_{++}^{②})$ in which both graphs are connected exists. It remains to show that it can be computed in linear time. To connect all the connected components of $G^{①}$, we contract each of them to a single vertex in the graph $G^{②}$. Then an arbitrary spanning tree yields a set of edges $e_1^{②}, \dots, e_k^{②} \in E^{②}$, such that augmenting the instance with respect to these edges yields a connected graph $G_{++}^{①}$. This works symmetrically for $G^{②}$ and can obviously be done in linear time. □

7.3 Disjoint Cycles

In this section, we consider the problem Sefe for the case that the common graph consists of a set of disjoint cycles. Due to Theorem 7.1, we can assume without loss of generality that both graphs are connected. In Section 7.3.1 we show how to solve this special case of Sefe in polynomial time. In Section 7.3.2 we introduce a tree-like data structure, the *CC-tree*, representing all planar embeddings of a set of cycles contained in a single graph that can be induced by an embedding of the whole graph. We additionally show that the intersection of the set of embeddings represented by two CC-trees can again be represented by a CC-tree, yielding a solution for Sefe even for the case of more than two graphs if all graphs have the same common graph, which consists of a set of disjoint cycles. In Section 7.3.3 we show how to compute the CC-tree and the intersection of two CC-trees in linear time. Before we start, we fix some definitions.

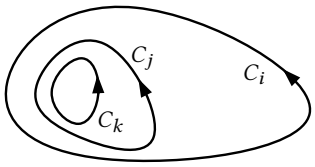


Figure 7.3: Three nested cycles. Their relative positions are $\text{pos}_{C_i}(C_j) = \text{pos}_{C_i}(C_k) = \text{pos}_{C_j}(C_k) =$ "left" and $\text{pos}_{C_k}(C_j) = \text{pos}_{C_k}(C_i) = \text{pos}_{C_j}(C_i) =$ "right". Changing for example $\text{pos}_{C_i}(C_k)$ to "right" without changing any other relative position is not possible.

Embeddings of Disjoint Cycles. Let $C = \{C_1, \ldots, C_k\}$ be a set of disjoint simple cycles. We consider embeddings of these cycles on the sphere. Since a single cycle has a unique embedding on the sphere only their relative positions to one another are of interest. To be able to use the terms "left" and "right" we consider the cycles to be directed. We denote the relative position of a cycle C_j with respect to a cycle C_i by $\text{pos}_{C_i}(C_j)$. More precisely, we have $\text{pos}_{C_i}(C_j) =$ "left" and $\text{pos}_{C_i}(C_j) =$ "right", if C_j lies on the left and right side of C_i, respectively. We call an assignment of a value "left" or "right" to each of these relative positions a *semi-embedding* of the cycles $C = \{C_1, \ldots, C_k\}$. Note that not every semi-embedding yields an embedding of the cycles. For example if $\text{pos}_{C_i}(C_j) = \text{pos}_{C_j}(C_k) =$ "left" and $\text{pos}_{C_j}(C_i) =$ "right", then $\text{pos}_{C_i}(C_k)$ also needs to have the value "left"; see Figure 7.3. However, two embeddings yielding the same semi-embedding are the same.

Sometimes we do not only consider the relative position of cycles but also of some other disjoint subgraph. We extend our notation to this case. For example the relative position of a single vertex v with respect to a cycle C is denoted by $\text{pos}_C(v)$.

7.3.1 A Polynomial-Time Algorithm

Let $(G^①, G^②)$ be an instance of Sefe with common graph G consisting of pairwise disjoint simple cycles $C = \{C_1, \ldots, C_k\}$. We first assume that $G^①$ and $G^②$ are biconnected and show later how to remove this restriction. Our approach is to formulate constraints on the relative positions of the cycles to one another ensuring that $G^①$ and $G^②$ induce the same semi-embedding on the common graph G. We show implicitly that the resulting semi-embedding is really an embedding by showing that the graphs $G^①$ and $G^②$ have planar embeddings inducing this semi-embedding. Note that this only works for the case that $G^①$ and $G^②$ are connected. Thus, our approach crucially relies on the result provided in Section 7.2.

Biconnected Graphs

Before considering two graphs, we determine for a single graph the possible embeddings it may induce on a set of disjoint cycles contained in it. Let $G = (V, E)$ be a biconnected graph with SPQR-tree $\mathcal{T}$, let C be a simple directed cycle in G and let μ be a node in $\mathcal{T}$. Obviously, C is either completely contained in the expansion graph

of a single virtual edge of μ or C induces a simple directed cycle of virtual edges in skel(μ). We say that C is *contracted* in skel(μ) in the first case and that C is *a cycle* in skel(μ) in the second case. If C is a cycle in skel(μ), we also say that skel(μ) *contains* C as a cycle. Consider the case where C is a cycle in skel(μ) and let κ denote this cycle. By fixing the embedding of skel(μ) the virtual edges in skel(μ) not contained in κ split into two groups, some lie to the left and some to the right of κ. Obviously, a vertex $v \in V \setminus V(C)$ in the expansion graph of a virtual edge that lies to the left (to the right) of κ lies to the left (to the right) of C in G, no matter which embedding is chosen for the skeletons of other nodes. In other words, the value of $\text{pos}_C(v)$ is completely determined by this single node μ. We show that for every vertex $v \in V \setminus V(C)$ there is a node μ in $\mathcal{T}$ containing C as a cycle such that the virtual edge in skel(μ) containing v in its expansion graph is not contained in the cycle κ induced by C. Hence such a node $\mu \in \mathcal{T}$ determining $\text{pos}_C(v)$ always exists. Extending this to a pair of cycles yields the following lemma.

Lemma 7.2. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ and let C_1 and C_2 be two disjoint simple cycles in G. There is exactly one node μ in $\mathcal{T}$ determining $\text{pos}_{C_1}(C_2)$. Moreover, μ contains C_1 as cycle κ_1 and C_2 either as a cycle or contracted in an edge not contained in κ_1.*

Proof. We choose some vertex $v \in V(C_2)$ as representative for the whole cycle. Consider a Q-node μ_1 in the SPQR-tree $\mathcal{T}$ corresponding to an edge contained in C_1. Moreover, let μ_k be a Q-node corresponding to an edge incident to v. We claim that the desired node μ lies somewhere on the path $\mu_1, \ldots, \mu_k$ in the SPQR-tree $\mathcal{T}$.

Obviously C_1 is a cycle in μ_1 and the vertex v belongs to the virtual edge in skel(μ_1). In μ_k the vertex v is a pole and C_1 is contracted in the virtual edge of skel(μ_k) since $v \notin V(C_1)$. Assume we are navigating from μ_1 to μ_k and let μ_i be the current node. If skel(μ_i) does not contain the vertex v, it belongs to a single virtual edge in skel(μ_i). In this case μ_{i+1} is obviously the node corresponding to this virtual edge. If v is a vertex of skel(μ_i), then μ_{i+1} corresponds to one of the virtual edges incident to v in skel(μ_i). As long as C_1 is a cycle in the current node and v belongs to a virtual edge in this cycle, the next node in the path corresponds to this virtual edge and thus C_1 remains a cycle. Since C_1 is contracted in μ_k, we somewhere need to follow a virtual edge not contained in the cycle induced by C_1; let μ be this node. By definition μ contains C_1 as cycle κ and the next node on the path belongs to a virtual edge that is not contained in κ but contains v in its expansion graph. Thus $\text{pos}_{C_1}(v)$ is determined by this node μ. Since v is a node of the second cycle C_2 also $\text{pos}_{C_1}(C_2)$ is completely determined by this node. Moreover, μ contains C_1 as cycle κ and C_2 either as a cycle or contracted in a virtual edge not belonging to κ. □

Now consider a set of pairwise disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ in G. Let μ be an arbitrary node in the SPQR-tree $\mathcal{T}$. If μ is an S- or a Q-node it clearly does not

determine any of the relative positions since either every cycle is contracted in skel(μ) or a single cycle is a cycle in skel(μ) containing all the virtual edges. In the following, we consider the two interesting cases namely that μ is an R- or a P-node containing at least one cycle as a cycle.

Let μ be a **P-node** in $\mathcal{T}$ with skel(μ) consisting of two vertices s and t with parallel virtual edges $\varepsilon_1, \ldots, \varepsilon_\ell$ between them. If $C \in \mathcal{C}$ is contained as a cycle in skel(μ), it induces a cycle κ in skel(μ) consisting of two of the parallel virtual edges. Let without loss of generality ε_1 and ε_2 be these virtual edges. Obviously, no other cycle $C' \in \mathcal{C}$ is a cycle in skel(μ) since such a cycle would need to contain s and t, which is a contradiction to the assumption that C and C' are disjoint. Thus, every other cycle C' is contracted in skel(μ), belonging to one of the virtual edges $\varepsilon_1, \ldots, \varepsilon_\ell$. If it belongs to ε_1 or ε_2, which are contained in κ, then $\text{pos}_C(C')$ is not determined by μ. If C' belongs to one of the virtual edges $\varepsilon_3, \ldots, \varepsilon_\ell$, the relative position $\text{pos}_C(C')$ is determined by the relative position of this virtual edge with respect to the cycle κ. This relative position can be chosen for every virtual edge $\varepsilon_3, \ldots, \varepsilon_\ell$ arbitrarily and independently. Hence, if there are two cycles C_i and C_j belonging to different virtual edges in μ, the positions $\text{pos}_C(C_i)$ and $\text{pos}_C(C_j)$ can be chosen independently. Furthermore, if the two cycles C_i and C_j belong to the same virtual edge $\varepsilon \in \{\varepsilon_3, \ldots, \varepsilon_\ell\}$, their relative position with respect to C is the same, i.e., $\text{pos}_C(C_i) = \text{pos}_C(C_j)$, for every embedding of G.

Let μ be an **R-node** in $\mathcal{T}$. For the moment, we consider that the embedding of skel(μ) is fixed by choosing one of the two orientations. Let C be a cycle inducing the cycle κ in skel(μ). Then the relative position $\text{pos}_C(C')$ of a cycle $C' \neq C$ is determined by μ if and only if C' is a cycle in skel(μ) or if it is contracted belonging to a virtual edge not contained in κ. Since we consider only one of the two embeddings of skel(μ) at the moment, $\text{pos}_C(C')$ is fixed to one of the two values "left" or "right" in this case. The same can be done for all other cycles that are cycles in skel(μ) yielding a fixed value for all relative positions that are determined by μ. Finally, we have a partition of all positions determined by μ into the set of positions $\mathcal{P}_1 = \{\text{pos}_{C_{a(1)}}(C_{b(1)}), \ldots, \text{pos}_{C_{a(r)}}(C_{b(r)})\}$ all having the value "left" and the set of positions $\mathcal{P}_2 = \{\text{pos}_{C_{c(1)}}(C_{d(1)}), \ldots, \text{pos}_{C_{c(s)}}(C_{d(s)})\}$ having the value "right". Now if the embedding of skel(μ) is not fixed anymore, we have only the possibility to flip it. By flipping, all the positions in $\mathcal{P}_1$ change to "right" and all positions in $\mathcal{P}_2$ change to "left". Hence, we obtain that the equation $\text{pos}_{C_{a(1)}}(C_{b(1)}) = \cdots = \text{pos}_{C_{a(r)}}(C_{b(r)}) \neq \text{pos}_{C_{c(1)}}(C_{d(1)}) = \cdots = \text{pos}_{C_{c(s)}}(C_{d(s)})$ is satisfied for every embedding of the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ induced by an embedding of G.

To sum up, we obtain a set of (in)equalities relating the relative positions of cycles to one another. We call these constraints the *PR-node constraints* with respect to the biconnected graph G. Obviously the PR-node constraints are necessary in the sense that every embedding of G induces an embedding of the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$

satisfying these constraints. The following lemma additionally states the sufficiency of the PR-node constraints.

Lemma 7.3. *Let G be a biconnected planar graph containing the disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. Let further $\mathcal{E}_\mathcal{C}$ be a semi-embedding of these cycles. There is an embedding $\mathcal{E}$ of G inducing $\mathcal{E}_\mathcal{C}$ if and only if $\mathcal{E}_\mathcal{C}$ satisfies the PR-node constraints.*

Proof. The "only if"-part of the proof is obvious, as mentioned above. It remains to show the "if"-part. Let $\mathcal{E}_\mathcal{C}$ be a semi-embedding of $\mathcal{C} = \{C_1, \ldots, C_k\}$ satisfying the PR-node constraints given by G. We show how to construct an embedding $\mathcal{E}$ of G inducing the embedding $\mathcal{E}_\mathcal{C}$ on the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. We simply process the nodes of the SPQR-tree one by one and choose an embedding for the skeleton of every node. Let μ be a node in $\mathcal{T}$. If μ is an S- or a Q-node, there is nothing to do, since there is no choice for the embedding of $\text{skel}(\mu)$. If μ is a P-node several relative positions may be determined by the embedding of $\text{skel}(\mu)$. However, these positions satisfy the PR-node constraints stemming from μ, hence we can choose an embedding for $\text{skel}(\mu)$ determining these positions as given by $\mathcal{E}_\mathcal{C}$. Obviously, the same holds for the case where μ is an R-node. Hence, we finally obtain an embedding $\mathcal{E}$ of G determining the positions that are determined by a node in $\mathcal{T}$ as required by $\mathcal{E}_\mathcal{C}$. Due to Lemma 7.2 every pair of relative positions is determined by exactly one node in $\mathcal{T}$, yielding that the resulting embedding $\mathcal{E}$ induces $\mathcal{E}_\mathcal{C}$ on the cycles. Note that this shows implicitly that $\mathcal{E}_\mathcal{C}$ is not only a semi-embedding but also an embedding. □

Now let $G^①$ and $G^②$ be two biconnected planar graphs with the common graph G consisting of pairwise disjoint simple cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. If we find a semi-embedding $\mathcal{E}$ of the cycles that satisfies the PR-node constraints with respect to $G^①$ and $G^②$ simultaneously, we can use Lemma 7.3 to find embeddings $\mathcal{E}^①$ and $\mathcal{E}^②$ for $G^①$ and $G^②$ both inducing the embedding $\mathcal{E}$ on the common graph G. Thus, satisfying the PR-node constraints with respect to both $G^①$ and $G^②$, is sufficient to find a simultaneous embedding. Conversely, given a pair of embeddings $\mathcal{E}^①$ and $\mathcal{E}^②$ inducing the same embedding $\mathcal{E}$ on G, this embedding $\mathcal{E}$ needs to satisfy the PR-node constraints with respect to both, $G^①$ and $G^②$, which is again due to Lemma 7.3. Since the PR-node constraints form a set of boolean (in)equalities we can express them as an instance of 2-Sat. As this instance has polynomial size and can easily be computed in polynomial time, we obtain the following theorem.

Theorem 7.2. *Sefe can be solved in quadratic time for biconnected graphs whose common graph is a set of disjoint cycles.*

Proof. It remains to show that the PR-node constraints can be computed in quadratic time, yielding an instance of 2-Sat with quadratic size. As this 2-Sat instance can be solved consuming time linear in its size [EIS76; APT79], we obtain a quadratic-time algorithm.

We show how to process each node μ of the SPQR-tree in $O(n \cdot |\operatorname{skel}(\mu)|)$ time, computing the PR-node constraints stemming from μ. For each virtual edge ε we compute a list of cycles in C that contain edges in the expansion graph $\operatorname{expan}(\varepsilon)$ by traversing all leaves in the corresponding subtree, consuming $O(n)$ time for each virtual edge. Then the list of cycles that occur as cycles in $\operatorname{skel}(\mu)$ can be computed in linear time. For each of these cycles C all constraints on relative positions with respect to C determined by μ can be easily computed in $O(n)$ time. As only $O(|\operatorname{skel}(\mu)|)$ cycles can be contained as cycles in $\operatorname{skel}(\mu)$, this yields the claimed $O(n \cdot |\operatorname{skel}(\mu)|)$ time for each skeleton. Since the total size of the skeletons is linear in the size of the graph, this yields an overall $O(n^2)$-time algorithm. □

Allowing Cutvertices

In this section we consider the case where the graphs may contain cutvertices. As before, we consider a single graph G containing a set of disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ first. Let $C \in \mathcal{C}$ be one of the cycles and let v be a cutvertex contained in the same block B that contains C. The cutvertex v splits G into ℓ cut components $H_1, \ldots, H_\ell$. Assume without loss of generality that B (and with it also C) is contained in H_1. We distinguish between the cases that v is contained in C and that it is not.

If ***v* is not contained in *C***, then the relative position $\operatorname{pos}_C(v)$ is determined by the embedding of the block B and it follows that all the subgraphs $H_2, \ldots, H_\ell$ lie on the same side of C as v does. It follows from the biconnected case (Lemma 7.2) that $\operatorname{pos}_C(v)$ is determined by the embedding of the skeleton of exactly one node μ in the SPQR-tree of B. Obviously, the conditions that all cycles in $H_2, \ldots, H_\ell$ are on the same side of C as v can be easily added to the PR-node constraints stemming from the node μ; call the resulting constraints the *extended PR-node constraints*. These constraints are clearly necessary. On the other hand, if $\mathcal{E}_C$ is a semi-embedding of the cycles satisfying the extended PR-node constraints, we can find an embedding $\mathcal{E}_B$ of the block B such that all relative positions of cycles that are determined by single nodes in the SPQR-tree of B are compatible with $\mathcal{E}_C$.

If ***v* is contained in *C***, the relative position $\operatorname{pos}_C(v)$ does not exist. Assume the embedding of each block is already chosen. Then for each subgraph $H \in \{H_2, \ldots, H_\ell\}$, the positions $\operatorname{pos}_C(H)$ can be chosen arbitrarily and independently. In this case we say for a cycle C' in H that its relative position $\operatorname{pos}_C(C')$ is *determined by the embedding chosen for the cutvertex v*. Obviously, in every embedding of G, a pair of cycles C_i and C_j both belonging to the same subgraph $H \in \{H_2, \ldots, H_\ell\}$ lie on the same side of C yielding the equation $\operatorname{pos}_C(C_i) = \operatorname{pos}_C(C_j)$. This equation can be set up for every pair of cycles in each of the subgraphs, yielding the *cutvertex constraints* with respect to v. Again we have that, given a semi-embedding $\mathcal{E}_C$ of the cycles satisfying the cutvertex constraints with respect to v, we can simply choose an embedding of the graph such

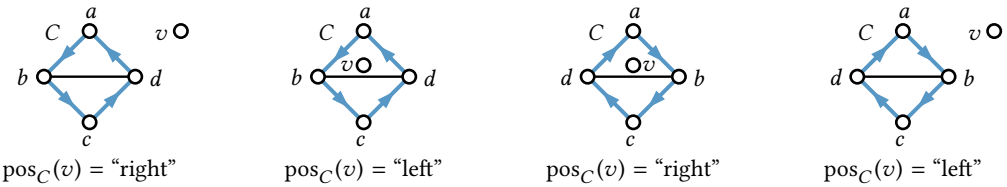


Figure 7.4: One component containing C (blue) and another consisting only of the vertex v. Changing the face in which v lies may change the relative position $\text{pos}_C(v)$. Moreover, changing the embedding of the component containing C (in this case flipping it) also changes $\text{pos}_C(v)$.

that the relative positions determined by the embedding around the cutvertex are compatible with $\mathcal{E}_C$.

To sum up, a semi-embedding $\mathcal{E}_C$ on the cycles $C = \{C_1, \ldots, C_k\}$ that is induced by an embedding $\mathcal{E}$ of the whole graph always satisfies the extended PR-node and cutvertex constraints. Moreover, given a semi-embedding $\mathcal{E}_C$ satisfying these constraints, we can find an embedding $\mathcal{E}$ of G inducing compatible relative positions for each relative position that is determined by a single node in the SPQR-tree of a block or by a cutvertex. Obviously, the relative position of every pair of cycles is determined by such a node or a cutvertex. Thus the extended PR-node and cutvertex constraints together are sufficient, i.e., given a semi-embedding of the cycles satisfying these constraints, we can find an embedding of G inducing this semi-embedding. This shows implicitly that the given semi-embedding is an embedding. This result is stated again in the following lemma.

Lemma 7.4. *Let G be a connected planar graph containing the disjoint cycles $C = \{C_1, \ldots, C_k\}$. Let further $\mathcal{E}_C$ be a semi-embedding of these cycles. There is an embedding $\mathcal{E}$ of G inducing $\mathcal{E}_C$ if and only if $\mathcal{E}_C$ satisfies the extended PR-node and cutvertex constraints with respect to G.*

This result again directly yields a polynomial-time algorithm to solve Sefe for the case that both graphs $G^{①}$ and $G^{②}$ are connected and their common graph G consists of a set of disjoint cycles. Moreover, requiring both graphs to be connected is not really a restriction due to Theorem 7.1. The extended PR-node and cutvertex constraints can be computed similarly as in the proof of Theorem 7.2, yielding the following theorem.

Theorem 7.3. *Sefe can be solved in quadratic time if the common graph consists of disjoint cycles.*

Note that we really need to use Theorem 7.1 to ensure that the graphs are connected since our approach does not extend to the case where the graphs are allowed to be disconnected. In this case it would still be easy to formulate necessary conditions in terms of boolean equations. However, these conditions would only be sufficient if it is

additionally ensured that the given semi-embedding actually is an embedding. The reason why this is not directly ensured by the embedding of the graph (as it is in the connected case) is that the relative position of cycles to one another is not determined by exactly one choice that can be made independently from the other choices; see Figure 7.4.

7.3.2 A Compact Representation of all Simultaneous Embeddings

In the previous section we showed that SEFE can be solved in polynomial time for the case that the common graph consists of disjoint cycles. In this section we describe a data structure, the *CC-tree*, representing all embeddings of a set of disjoint cycles that can be induced by an embedding of a connected graph containing them. Afterwards, we show that the intersection of the sets of embeddings represented by two CC-trees can again be represented by a CC-tree. In Section 7.3.3 we then show that the CC-tree and the intersection of two CC-trees can be computed in linear time, yielding an optimal linear-time algorithm for SEFE for the case that the common graph consists of disjoint cycles. Note that this algorithm obviously extends to the case where k graphs $G^{①},\dots,G^{ⓚ}$ are given such that they all intersect in the same common graph G consisting of a set of disjoint cycles.

C-Trees and CC-Trees

Let $\mathcal{C} = \{C_1,\dots,C_k\}$ be a set of disjoint cycles. A *cycle-tree (C-tree)* $\mathcal{T}_\mathcal{C}$ on these cycles is a minimal connected graph containing $\mathcal{C}$; see Figure 7.5. Obviously, every embedding of $\mathcal{T}_\mathcal{C}$ induces an embedding of the cycles. We say that two embeddings of $\mathcal{T}_\mathcal{C}$ are equivalent if they induce the same embedding of $\mathcal{C}$ and we are only interested in the equivalence classes with respect to this equivalence relation. An embedding $\mathcal{E}$ of the cycles in $\mathcal{C}$ is *represented* by $\mathcal{T}_\mathcal{C}$ if it admits an embedding inducing $\mathcal{E}$. Note that contracting each of the cycles $\mathcal{C} = \{C_1,\dots,C_k\}$ in a C-tree to a single vertex yields a spanning tree on these vertices. In most cases we implicitly assume the cycles to be contracted such that $\mathcal{T}_\mathcal{C}$ can be treated like a tree.

The embedding choices that can be made for $\mathcal{T}_\mathcal{C}$ are of the following kind. For every edge $e = \{C,C'\}$ in $\mathcal{T}_\mathcal{C}$, we can decide to put all cycles in the subtree attached to C via e either to the left or to the right of C. In particular, we can assign a value "left" or "right" to the relative position $\text{pos}_C(C')$. Moreover, by fixing the relative positions $\text{pos}_C(C')$ and $\text{pos}_{C'}(C)$ for every pair of cycles C and C' that are adjacent in $\mathcal{T}_\mathcal{C}$, the embedding represented by $\mathcal{T}_\mathcal{C}$ is completely determined. Thus, given a C-tree $\mathcal{T}_\mathcal{C}$, we call the relative positions $\text{pos}_C(C')$ and $\text{pos}_{C'}(C)$ with $C,C' \in \mathcal{C}$ *crucial* if C and C' are adjacent in $\mathcal{T}_\mathcal{C}$; see Figure 7.5. We note that, when determining an embedding of $\mathcal{T}_\mathcal{C}$, the crucial relative positions can be chosen independently from one another.

Since the crucial relative positions with respect to a C-tree $\mathcal{T}_\mathcal{C}$ are binary variables,

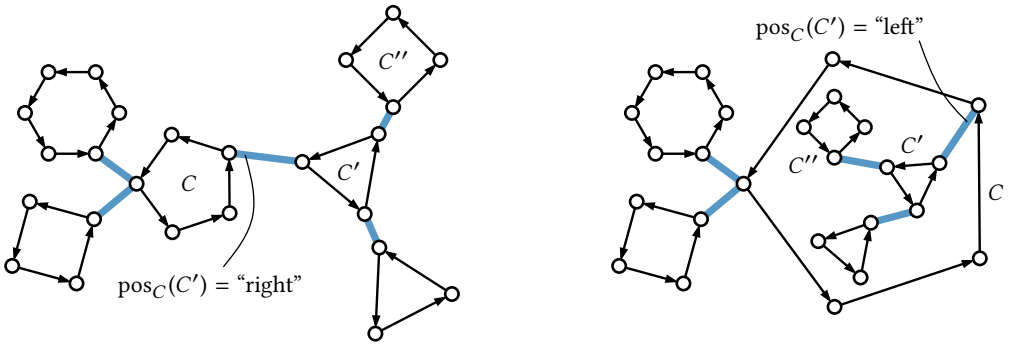


Figure 7.5: Two embeddings of the same CC-tree. The only difference between the embeddings is that different values are chosen for the crucial relative position $\text{pos}_C(C')$. Note that the tree structure enforces the (non-crucial) relative position $\text{pos}_C(C'')$ to be equal to $\text{pos}_C(C')$.

we can use (in)equalities between them to further constrain the embeddings represented by $\mathcal{T}_C$. We call a C-tree with such additional constraints on its crucial relative positions a *constrained cycle-tree (CC-tree)* on the set of cycles C. In this way, there is a bijection between the embeddings of C represented by a CC-tree and the solutions of an instance of 2-Sat given by the constraints on the crucial relative positions of $\mathcal{T}_C$. We essentially prove two things. First, for every connected graph G containing the cycles C, there exists a CC-tree representing exactly the embeddings of C that can be induced by embeddings of G. Essentially, we have to restrict the extended PR-node and cutvertex constraints to the crucial relative positions of a C-tree compatible with G. Second, for a pair of CC-trees $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$ on the same set C of cycles, there exists a CC-tree $\mathcal{T}_C$ representing exactly the embeddings of C that are represented by $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$.

Let G be a connected planar graph containing a set C of disjoint cycles. We say that a C-tree $\mathcal{T}_C$ is *compatible* with G if it is a minor of G, i.e., if it can be obtained by contracting edges in a subgraph of G. The corresponding *compatible CC-tree* is obtained from $\mathcal{T}_C$ by adding the subset of the extended PR-node and cutvertex constraints that only involve crucial relative positions of $\mathcal{T}_C$. Note that there may be many compatible CC-trees for a single graph G. However, in the following we arbitrarily fix one of them and speak about *the* CC-tree of G.

Theorem 7.4. *Let G be a connected planar graph containing the disjoint cycles $C = \{C_1, \ldots, C_k\}$. The CC-tree $\mathcal{T}_C$ of G represents exactly the embeddings of C that can be induced by an embedding of G.*

Proof. Let $\mathcal{E}$ be an embedding of G and let $\mathcal{E}_C$ be the embedding induced on the cycles $C = \{C_1, \ldots, C_k\}$. Obviously, the CC-tree $\mathcal{T}_C$ can be obtained from G by contracting the cycles C to single vertices, choosing a spanning tree, expanding the

cycles and contracting edges incident to non-cycle vertices. Since we essentially only pick a subgraph of G containing all cycles $\mathcal{C} = \{C_1, \dots, C_k\}$ and contract edges, the embedding $\mathcal{E}_C$ is preserved. Moreover, by Lemma 7.4, it satisfies the extended PR-node and cutvertex constraints since it is induced by the embedding $\mathcal{E}$ of G. Hence, $\mathcal{E}_C$ is represented by the CC-tree $\mathcal{T}_C$.

Conversely, let $\mathcal{E}_C$ be an embedding on the cycles represented by the CC-tree $\mathcal{T}_C$. By definition, the extended PR-node and cutvertex constraints are satisfied for the crucial relative positions. We show that the tree-like structure of $\mathcal{T}_C$ ensures that they are also satisfied for the remaining relative positions, yielding that an embedding $\mathcal{E}$ of G inducing $\mathcal{E}_C$ exists due to Lemma 7.4.

We start with the PR-node constraints. Let B be a block of G with SPQR-tree $\mathcal{T}(B)$. In a P-node μ containing a cycle C as cycle κ every other cycle in B is contracted, belonging to a single virtual edge. Let C_i and C_j be two cycles in B belonging to the same virtual edge ε not contained in κ. In this case the PR-node constraints stemming from μ require $\text{pos}_C(C_i) = \text{pos}_C(C_j)$, and we show that this equation is implied if the extended PR-node constraints are satisfied for the crucial relative positions. Let C_i' and C_j' be the first cycles on the paths from C to C_i and C_j in $\mathcal{T}_C$, respectively. Note that C_i' and C_j' are not necessarily contained in the block B. However, we first consider the case where both are contained in B. Then C_i' and C_j' are both contracted in the same virtual edge ε as C_i and C_j since a path from a cycle belonging to ε to a cycle belonging to a different virtual edge would necessarily contain a pole of $\text{skel}(\mu)$ and thus a vertex in C. Thus, the PR-node constraints restricted to the crucial relative positions enforce $\text{pos}_C(C_i') = \text{pos}_C(C_j')$. Furthermore, the tree structure of $\mathcal{T}_C$ enforces $\text{pos}_C(C_i) = \text{pos}_C(C_i')$ and $\text{pos}_C(C_j) = \text{pos}_C(C_j')$. Hence, in this case the PR-node constraints stemming from μ are implied by their restriction to the crucial relative positions.

For the case that C_i' or C_j' are contained in a different block, they are connected to B via cutvertices v_i or v_j, which must belong to $\text{expan}(\varepsilon)$ by the same argument as above, namely that every path from C_i or C_j to a vertex that is contained in the expansion graph of another virtual edge needs to contain one of the poles. Thus, the extended PR-node constraints enforce $\text{pos}_C(C_i') = \text{pos}_C(C_j')$ yielding the same situation as above. In total, the extended PR-node constraints stemming from a P-node μ restricted to the crucial relative positions enforce that the PR-node constraints stemming from μ are satisfied for all relative positions.

For the case that μ is an R-node a similar argument holds. If C is a cycle κ in $\text{skel}(\mu)$ and two cycles C_i and C_j lie contracted or as cycles on the same side (on different sides) of κ, then the first cycles C_i' and C_j' on the path from C to C_i and C_j in the CC-tree $\mathcal{T}_C$ lie on the same side (on different sides) of κ or the cutvertices connecting C_i' and C_j' to the block B lie on the same side (on different sides) of κ. Thus, the extended PR-node constraints restricted to the crucial relative positions enforce $\text{pos}_C(C_i') = \text{pos}_C(C_j')$

($\text{pos}_C(C_i') \neq \text{pos}_C(C_j')$) and the tree structure of $\mathcal{T}_C$ yields $\text{pos}_C(C_i) = \text{pos}_C(C_i')$ and $\text{pos}_C(C_j) = \text{pos}_C(C_j')$. Obviously, these arguments extend to the case of extended PR-node constraints since a cutvertex not contained in C can be treated like a disjoint cycle.

It remains to deal with the cutvertex constraints stemming from the case where C is a cycle containing a cutvertex v splitting G into the cut components $H_1, \ldots, H_\ell$. Let without loss of generality H_1 be the subgraph containing C. The cutvertex constraints ensure that a pair of cycles C_i and C_j belonging to the same subgraph $H \in \{H_2, \ldots, H_\ell\}$ are located on the same side of C. Let C_i' and C_j' be the first cycles on the path from C to C_i and C_j in the CC-tree $\mathcal{T}_C$, respectively. Obviously C_i' and C_j' belong to the same subgraph H and hence the cutvertex constraints restricted to the crucial relative positions enforce $\text{pos}_C(C_i') = \text{pos}_C(C_j')$. Moreover, the tree structure of $\mathcal{T}_C$ again ensures that the equations $\text{pos}_C(C_i) = \text{pos}_C(C_i')$ and $\text{pos}_C(C_j) = \text{pos}_C(C_j')$ hold, which concludes the proof. □

Intersecting CC-Trees

In this section we consider two CC-trees $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$ on the same set of cycles $\mathcal{C}$. We show that the set of embeddings that are represented by both $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$ can again be represented by a single CC-tree. We will show this by constructing a new CC-tree, which we call the *intersection* of $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$, showing afterwards that this CC-tree has the desired property. The intersection $\mathcal{T}_C$ is a copy of $\mathcal{T}_C^{①}$ with some additional constraints given by the second CC-tree $\mathcal{T}_C^{②}$. We essentially have to formulate two types of constraints. First, constraints stemming from the structure of the underlying C-tree of $\mathcal{T}_C^{②}$. Second, the constraints given by the (in)equalities on the relative positions that are crucial with respect to $\mathcal{T}_C^{②}$. We show that both kinds of constraints can be formulated as (in)equalities on the relative positions that are crucial with respect to $\mathcal{T}_C^{①}$.

Let C_1 and C_2 be two cycles joined by an edge in $\mathcal{T}_C^{②}$. Obviously, C_1 and C_2 are contained in the boundary of a common face in every embedding $\mathcal{E}^{②}$ represented by $\mathcal{T}_C^{②}$. It is easy to formulate constraints on the relative positions that are crucial with respect to $\mathcal{T}_C^{①}$ such that C_1 and C_2 are contained in the boundary of a common face for every embedding represented by $\mathcal{T}_C^{①}$. Consider the path π from C_1 to C_2 in $\mathcal{T}_C^{①}$. For every three cycles C, C' and C'' appearing consecutively on π it is necessary that $\text{pos}_{C'}(C) = \text{pos}_{C'}(C'')$ holds. Otherwise C_1 and C_2 would be separated by C'. Conversely, if this equation holds for every triple of consecutive cycles on π, then C_1 and C_2 always lie on a common face. We call the resulting equations the *common-face constraints*. Note that all relative positions involved in such constraints are crucial with respect to $\mathcal{T}_C^{①}$.

To formulate the constraints given on the crucial relative positions of $\mathcal{T}_C^{②}$, we

essentially find, for each of these crucial relative positions $\mathrm{pos}_{C_1}(C_2)$, a relative position $\mathrm{pos}_{C_1}(C_2')$ that is crucial with respect to $\mathcal{T}_C^{①}$ such that $\mathrm{pos}_{C_1}(C_2)$ is determined by fixing $\mathrm{pos}_{C_1}(C_2')$ in $\mathcal{T}_C^{①}$. More precisely, for every relative position $\mathrm{pos}_{C_1}(C_2)$ that is crucial with respect to $\mathcal{T}_C^{②}$ we define its *representative* in $\mathcal{T}_C^{①}$ to be the crucial relative position $\mathrm{pos}_{C_1}(C_2')$, where C_2' is the first cycle in $\mathcal{T}_C^{①}$ on the path from C_1 to C_2. We obtain the *crucial-position constraints* on the crucial relative positions of $\mathcal{T}_C^{①}$ by replacing every relative position in the constraints given for $\mathcal{T}_C^{②}$ by its representative. The resulting set of (in)equalities on the crucial relative positions of $\mathcal{T}_C^{①}$ is obviously necessary.

We can now formally define the *intersection* $\mathcal{T}_C$ of two CC-trees $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$ to be $\mathcal{T}_C^{①}$ with the common-face and crucial-position constraints additionally restricting its crucial relative positions. We obtain the following theorem, justifying the name "intersection".

Theorem 7.5. *The intersection of two CC-trees represents exactly the embeddings that are represented by both CC-trees.*

Proof. Let $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$ be two CC-trees and let $\mathcal{T}_C$ be their intersection. Let further $\mathcal{E}$ be an embedding represented by $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$. Then $\mathcal{T}_C$ also represents $\mathcal{E}$ since the common-face and crucial-position constraints are obviously necessary. Now let $\mathcal{E}$ be an embedding represented by $\mathcal{T}_C$. It is clearly also represented by $\mathcal{T}_C^{①}$ since $\mathcal{T}_C$ is the same tree with some additional constraints. It remains to show that $\mathcal{E}$ is represented by $\mathcal{T}_C^{②}$. The embedding $\mathcal{E}$ induces a value for every relative position. In particular, it induces a value for every relative position that is crucial with respect to $\mathcal{T}_C^{②}$. The crucial-position constraints ensure that these values satisfy the constraints given for the crucial relative positions in the CC-tree $\mathcal{T}_C^{②}$. Thus we can simply take these positions, apply them to $\mathcal{T}_C^{②}$ and obtain an embedding $\mathcal{E}^{②}$ that is represented by $\mathcal{T}_C^{②}$. It remains to show that $\mathcal{E} = \mathcal{E}^{②}$. To this end, we consider an arbitrary pair of cycles C_1 and C_2 and show the following equation, where $\mathrm{pos}_{C_1}(C_2)$ and $\mathrm{pos}^{②}_{C_1}(C_2)$ denote the relative positions of C_2 with respect to C_1 in the embeddings $\mathcal{E}$ and $\mathcal{E}^{②}$, respectively.

$$\mathrm{pos}_{C_1}(C_2) \quad = \quad \mathrm{pos}^{②}_{C_1}(C_2) \tag{7.1}$$

Consider the paths π and $\pi^{②}$ from C_1 to C_2 in $\mathcal{T}_C$ and $\mathcal{T}_C^{②}$, respectively. We use induction on the length of $\pi^{②}$, illustrated in Figure 7.6, with Equation (7.1) as induction hypothesis. If $|\pi^{②}| = 1$, then $\mathrm{pos}_{C_1}(C_2)$ is crucial with respect to $\mathcal{T}_C^{②}$ and thus equal in both embeddings $\mathcal{E}$ and $\mathcal{E}^{②}$ by construction of $\mathcal{E}^{②}$. For the case $|\pi^{②}| > 1$ let C_1' and $C_1^{②}$ be the neighbors of C_1 in π and $\pi^{②}$, respectively. Since C_1' and $C_1^{②}$ lie on the path between C_1 and C_2 in $\mathcal{T}_C$ and $\mathcal{T}_C^{②}$, the following two equations hold.

$$\mathrm{pos}_{C_1}(C_2) \quad = \quad \mathrm{pos}_{C_1}(C_1') \tag{7.2}$$

$$\mathrm{pos}^{②}_{C_1}(C_2) \quad = \quad \mathrm{pos}^{②}_{C_1}(C_1^{②}) \tag{7.3}$$

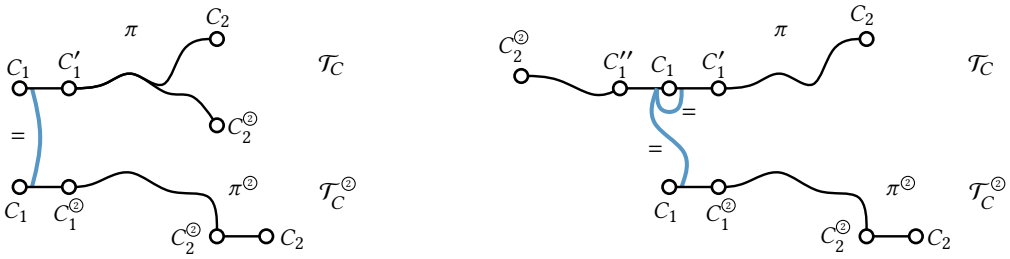


Figure 7.6: The two cases arising in the proof of Theorem 7.5. If the path from C_1 to $C_2^{②}$ starts with the edge $\{C_1, C_1'\}$ (left) the equation $\text{pos}_{C_1}(C_1') = \text{pos}^{②}_{C_1}(C_1^{②})$ follows by induction. Otherwise (right) $\text{pos}_{C_1}(C_1'') = \text{pos}^{②}_{C_1}(C_1^{②})$ follows by induction and the equation $\text{pos}_{C_1}(C_1') = \text{pos}_{C_1}(C_1'')$ holds due to the common-face constraint stemming from $\{C_2, C_2^{②}\}$.

Thus, it suffices to show that $\text{pos}_{C_1}(C_1') = \text{pos}^{②}_{C_1}(C_1^{②})$ holds to obtain Equation (7.1). Let $C_2^{②}$ be the neighbor of C_2 on the path $\pi^{②}$. Since the path from C_1 to $C_2^{②}$ is shorter than $\pi^{②}$ the equation $\text{pos}_{C_1}(C_2^{②}) = \text{pos}^{②}_{C_1}(C_2^{②})$ follows from the induction hypothesis stated in Equation (7.1). There are two possibilities. The path from C_1 to $C_2^{②}$ in $\mathcal{T}_C$ has either $\{C_1, C_1'\}$ or $\{C_1, C_1''\}$ for some other cycle C_1'' as first edge. In the former case the equation

$$\text{pos}_{C_1}(C_1') \quad = \quad \text{pos}^{②}_{C_1}(C_1^{②}) \tag{7.4}$$

obviously follows. Together with Equations (7.2) and (7.3), this yields the induction hypothesis (Equation (7.1)). In the latter case we have the following equation.

$$\text{pos}_{C_1}(C_1'') \quad = \quad \text{pos}^{②}_{C_1}(C_1^{②}) \tag{7.5}$$

Moreover, the common-face constraints stemming from the edge $\{C_2^{②}, C_2\}$ in $\mathcal{T}_C^{②}$ enforce

$$\text{pos}_{C_1}(C_1'') \quad = \quad \text{pos}_{C_1}(C_1'), \tag{7.6}$$

again yielding the induction hypothesis stated in Equation (7.1). This concludes the proof. □

7.3.3 Linear-Time Algorithm

In this section we first show how to compute the CC-tree of a given graph containing a set of disjoint cycles in linear time. Afterwards, we show that the intersection of two CC-trees can be computed in linear time. Together, this yields a linear-time algorithm for the variant of Sefe we consider.

Computing the CC-Tree in Linear Time

The first step of computing the CC-tree $\mathcal{T}_C$ of a graph G is to compute the underlying C-tree. Obviously, this can be easily done in linear time. Thus, the focus of this section lies on computing the extended PR-node and cutvertex constraints restricted to the crucial relative positions. To simplify notation we first consider the case where G is biconnected. Before we start computing the PR-node constraints we need one more definition. For each cycle C there is a set of inner nodes in the SPQR-tree $\mathcal{T}$ containing C as a cycle. We denote the subgraph of $\mathcal{T}$ induced by these nodes by $\mathcal{T}|_C$ and call it the *induced subtree* with respect to C. To justify the term "subtree" we prove the following lemma.

Lemma 7.5. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ containing the disjoint cycles $\mathcal{C} = \{C_1, \dots, C_k\}$. The induced subtrees $\mathcal{T}|_{C_1}, \dots, \mathcal{T}|_{C_k}$ with respect to $C_1, \dots, C_k$ are pairwise edge-disjoint trees.*

Proof. We first show that the induced tree with respect to a single cycle is really a tree. Afterwards, we show that two disjoint cycles induce edge-disjoint trees, yielding that they have linear size in total.

Let C be a cycle in G and let $\mathcal{T}|_C$ be its induced tree. A Q-node in $\mathcal{T}$ contains C as a cycle if and only if the corresponding edge is contained in C. For each pair of these Q-nodes all nodes on the path between them are contained in $\mathcal{T}|_C$, thus the Q-nodes are in the same connected component in the induced subtree. Moreover, an internal node in $\mathcal{T}$ cannot be a leaf in $\mathcal{T}|_C$, implying that it contains only one connected component.

Assume there are two cycles C_i and C_j inducing trees $\mathcal{T}|_{C_i}$ and $\mathcal{T}|_{C_j}$ that are not edge-disjoint. Let $\{\mu, \mu'\}$ be an edge in $\mathcal{T}$ belonging to both. Let further κ_i and κ_j be the cycles in skel(μ) induced by C_i and C_j, respectively. Since the neighbor μ' of μ also contains C_i as a cycle, it corresponds to a virtual edge ε in μ that is contained in κ_i. Similarly, ε is also contained in κ_j, which is a contradiction since C_i and C_j are disjoint. □

Our algorithm computing the PR-node constraints consists of four phases, each of them consuming linear time. In each phase we compute data we then use in the next phase. Table 7.1 gives an overview about the data we compute. During all phases we assume the SPQR-tree $\mathcal{T}$ to be rooted at a Q-node corresponding to an edge in G that is not contained in any cycle in $\mathcal{C}$. In the first phase we essentially compute the induced trees $\mathcal{T}|_C$. More precisely, for every node μ in the SPQR-tree we compute a list cyc(μ) containing a cycle C if and only if C is a cycle in skel(μ), i.e., if and only if μ is contained in $\mathcal{T}|_C$. Moreover, we say a virtual edge ε in skel(μ) *belongs* to a cycle C if C induces a cycle in skel(μ) containing ε. Note that ε belongs to at most one cycle. If ε belongs to C, we set bel(ε) = C; if ε does not belong to any cycle, we set

$\text{bel}(\varepsilon) = \perp$. Finally, the root of an induced tree $\mathcal{T}|_C$ with respect to the root chosen for $\mathcal{T}$ is denoted by $\text{root}(\mathcal{T}|_C)$. To sum up, in the first phase we compute $\text{cyc}(\mu)$ for every node μ, $\text{bel}(\varepsilon)$ for every virtual edge ε and $\text{root}(\mathcal{T}|_C)$ for every induced subtree $\mathcal{T}|_C$. In the second phase, we compute $\text{high}(\mu)$ as the highest edge in the SPQR-tree $\mathcal{T}$ on the path from μ to the root whose endpoints are both reachable from μ without using edges contained in any of the induced subtrees $\mathcal{T}|_C$. Note that $\text{high}(\mu)$ is the edge in $\mathcal{T}$ incident to the root if no edge on the path from μ to the root is contained in one of the induced subtrees. For the special case that the edge from μ to its parent itself is already contained in one of the induced trees, the edge $\text{high}(\mu)$ is not defined and we set $\text{high}(\mu) = \perp$. In the third phase we compute for every crucial relative position $\text{pos}_C(C')$ the node in the SPQR-tree determining it, denoted by $\text{det}(\text{pos}_C(C'))$. Moreover, for every virtual edge ε in $\text{skel}(\mu)$ we compute a list $\text{contr}(\varepsilon)$ of relative positions. A relative position $\text{pos}_C(C')$ is contained in $\text{contr}(\varepsilon)$ if and only if it is crucial, determined by μ and C' is contracted in ε. Similarly, the list $\text{detcyc}(\mu)$ for an R-node μ contains the crucial relative position $\text{pos}_C(C')$ if and only if C and C' are both cycles in $\text{skel}(\mu)$, implying that $\text{pos}_C(C')$ is determined by μ. Finally, in the fourth pase, we compute the PR-node constraints restricted to the crucial relative positions. The next lemma states that the first phase can be implemented in linear time.

Lemma 7.6. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ containing the disjoint cycles C. The data* $\text{cyc}(\mu)$ *for every node μ,* $\text{bel}(\varepsilon)$ *for every virtual edge ε, and* $\text{root}(\mathcal{T}|_{C_i})$ *for every cycle C_i can be computed in overall linear time.*

Proof. We process the SPQR-tree $\mathcal{T}$ bottom-up, starting with the Q-nodes. If a Q-node μ corresponds to an edge belonging to a cycle C, then $\text{cyc}(\mu)$ contains only C and $\text{bel}(\varepsilon) = C$ for the virtual edge in μ. If the edge corresponding to μ is not contained in a cycle, then $\text{cyc}(\mu)$ is empty and $\text{bel}(\varepsilon) = \perp$. Furthermore, a Q-node cannot be the root of any induced subtree $\mathcal{T}|_C$ as we chose as the root of $\mathcal{T}$ a Q-node corresponding to an edge not contained in any of the cycles. Now consider an inner node μ. We first process the virtual edges in $\text{skel}(\mu)$ not belonging to the parent of μ. Let ε be such a virtual edge corresponding to the child μ' of μ and let ε' be the virtual edge in $\text{skel}(\mu')$ corresponding to its parent μ. Then ε belongs to a cycle induced by C if and only if ε' does, thus we set $\text{bel}(\varepsilon) = \text{bel}(\varepsilon')$. Moreover, if $\text{bel}(\varepsilon) \neq \perp$ we need to add the cycle $\text{bel}(\varepsilon)$ to $\text{cyc}(\mu)$ if it was not already added. Whether $\text{bel}(\varepsilon)$ is already contained in $\text{cyc}(\mu)$ can be tested in constant time as follows. We define a timestamp t, increase t every time we go to the next node in $\mathcal{T}$ and we store the current value of t for a cycle added to $\text{cyc}(\mu)$. Then a cycle C was already added to $\text{cyc}(\mu)$ if and only if the timestamp stored for C is equal to the current timestamp t. Thus, processing all virtual edges in $\text{skel}(\mu)$ not corresponding to the parent of μ takes time linear in the size of $\text{skel}(\mu)$. Let now $\varepsilon = \{s, t\}$ be the virtual edge corresponding to the parent of μ. If $\text{bel}(\varepsilon') = \perp$ for all virtual edges ε' incident to s, then ε cannot be contained in a cycle induced by any of the cycles in C. Otherwise, there are two possibilities. There is a

Data	Description
$\mathrm{cyc}(\mu)$	For a node μ in the SPQR-tree the list of cycles in $\mathcal{C}$ that are cycles in $\mathrm{skel}(\mu)$.
$\mathrm{bel}(\varepsilon)$	For a virtual edge ε in $\mathrm{skel}(\mu)$ either a cycle $C \in \mathcal{C}$ if C induces a cycle in $\mathrm{skel}(\mu)$ containing ε or $\perp$ denoting that ε is not contained in such a cycle.
$\mathrm{root}(\mathcal{T}\|_C)$	The root for the induced tree $\mathcal{T}\|_C$ with respect to a chosen root for the SPQR-tree $\mathcal{T}$.
$\mathrm{high}(\mu)$	For a node μ in the SPQR-tree $\mathcal{T}$ the highest edge in $\mathcal{T}$ on the path from μ to the root that is reachable without using an edge in any of the induced subtrees $\mathcal{T}\|_C$.
$\mathrm{det}(\mathrm{pos}_C(C'))$	The node in the SPQR-tree determining the relative position $\mathrm{pos}_C(C')$ of the cycle C' with respect to another cycle C.
$\mathrm{contr}(\varepsilon)$	For a virtual edge ε in $\mathrm{skel}(\mu)$ a list of relative positions containing $\mathrm{pos}_C(C')$ if and only if it is crucial, determined by μ and C' is contracted in ε.
$\mathrm{detcyc}(\mu)$	For every R-node μ a list of crucial relative positions containing $\mathrm{pos}_C(C')$ if and only if C and C' are cycles in $\mathrm{skel}(\mu)$.

Table 7.1: Data that is computed to compute the PR-node constraints restricted to the crucial relative positions.

cycle $C \in \mathcal{C}$ such that $\mathrm{bel}(\varepsilon_1) = C$ for exactly one virtual edge ε_1 incident to s or there are two such edges ε_1 and ε_2 with $\mathrm{bel}(\varepsilon_1) = \mathrm{bel}(\varepsilon_2) = C$. In the former case the edges belonging to C in $\mathrm{skel}(\mu)$ form a path from s to t, thus the edge ε corresponding to the parent also belongs to C and we set $\mathrm{bel}(\varepsilon) = C$. In the latter case s is contained in the cycle C but the virtual edge does not belong to C and we set $\mathrm{bel}(\varepsilon) = \perp$. This takes time linear in the degree of s in $\mathrm{skel}(\mu)$ and hence lies in $O(|\,\mathrm{skel}(\mu)|)$. It remains to set $\mathrm{root}(\mathcal{T}|_C) = \mu$ for every cycle C inducing the subtree $\mathcal{T}|_C$ having μ as root. The tree $\mathcal{T}|_C$ has μ as root if and only if C is contained as cycle κ in μ but the virtual edge ε in $\mathrm{skel}(\mu)$ corresponding to the parent of μ is not contained in κ. Thus we have to set $\mathrm{root}(\mathcal{T}|_C) = \mu$ for all cycles C in $\mathrm{cyc}(\mu)$ except for $\mathrm{bel}(\varepsilon)$. Note that this again consumes time linear in the size of $\mathrm{skel}(\mu)$ since the number of cycles that are cycles in μ is in $O(|\,\mathrm{skel}(\mu)|)$. Due to the fact that the SPQR-tree $\mathcal{T}$ has linear size this yields an overall linear running time. □

In the second phase we want to compute $\mathrm{high}(\mu)$ for each of the nodes in $\mathcal{T}$. We obtain the following lemma.

Lemma 7.7. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ containing the disjoint cycles $\mathcal{C}$. For every node μ in $\mathcal{T}$ the edge* $\text{high}(\mu)$ *can be computed in linear time.*

Proof. We make use of the fact that $\text{bel}(\varepsilon)$ is already computed for every virtual edge ε in each of the skeletons, which can be done in linear time due to Lemma 7.6. Note that an edge $\{\mu, \mu'\}$ in the SPQR-tree $\mathcal{T}$ (where μ is the parent of μ') belongs to the induced subtree $\mathcal{T}|_C$ with respect to the cycle $C \in \mathcal{C}$ if and only if $\text{bel}(\varepsilon) = C$ for the virtual edge ε in $\text{skel}(\mu)$ corresponding to the child μ'. In this case we also have $\text{bel}(\varepsilon') = \text{bel}(\varepsilon) = C$ where ε' is the virtual edge in $\text{skel}(\mu')$ corresponding to the parent. Hence, we can compute $\text{high}(\mu)$ for every node μ in $\mathcal{T}$ by processing $\mathcal{T}$ top-down remembering the latest processed edge not belonging to any of the induced subtrees. This can easily be done in linear time. □

In the third phase we compute $\det(\text{pos}_C(C'))$ for every crucial relative position in linear time. Moreover, we compute $\text{contr}(\varepsilon)$ for every virtual edge ε and $\text{detcyc}(\mu)$ for every R-node μ. We show the following lemma.

Lemma 7.8. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ containing the disjoint cycles $\mathcal{C}$. The node* $\det(\text{pos}_C(C'))$ *for each crucial relative position* $\text{pos}_C(C')$*, the list* $\text{contr}(\varepsilon)$ *for each virtual edges ε and the list* $\text{detcyc}(\mu)$ *for each R-nodes μ can be computed in overall linear time.*

Proof. Let C and C' be two cycles such that $\text{pos}_C(C')$ is a crucial relative position. We show how to compute the node $\det(\text{pos}_C(C'))$ determining this relative position in constant time. Moreover, if C' is contracted in a virtual edge ε in $\det(\text{pos}_C(C'))$, we append the relative position $\text{pos}_C(C')$ to $\text{contr}(\varepsilon)$. Otherwise, $\det(\text{pos}_C(C'))$ is an R-node containing C and C' as cycles and we add $\text{pos}_C(C')$ to $\text{detcyc}(\det(\text{pos}_C(C')))$. Since there are only linearly many crucial relative positions this takes only linear time. Let $\mu = \text{root}(\mathcal{T}|_C)$ and $\mu' = \text{root}(\mathcal{T}|_{C'})$ be the roots of the induced trees with respect to C and C', respectively, which are already computed due to Lemma 7.6. We use that the lowest common ancestor of a pair of nodes can be computed in constant time after a linear-time preprocessing [HT84; BF00]. In particular, let $\text{LCA}(\mu, \mu')$ be the lowest common ancestor of the two roots. There are three possibilities. First, $\text{LCA}(\mu, \mu')$ is above μ (Figure 7.7a). Second, $\text{LCA}(\mu, \mu') = \mu = \mu'$ (Figure 7.7b). And third, $\text{LCA}(\mu, \mu') = \mu$ lies above μ' (Figure 7.7c–f). Note that the first case includes the situation where $\mu' = \text{LCA}(\mu, \mu')$ lies above μ.

In the first case the cycle C' is contracted in μ in the virtual edge ε corresponding to the parent of μ, while μ contains C as cycle κ not containing the virtual edge ε corresponding to the parent. Hence, μ determines $\text{pos}_C(C')$. We set $\det(\text{pos}_C(C')) = \mu$ and insert $\text{pos}_C(C')$ into $\text{contr}(\varepsilon)$. In the second case C and C' are both cycles in $\mu = \mu'$, hence μ determines $\text{pos}_C(C')$. We set $\det(\text{pos}_C(C')) = \mu$ and insert $\text{pos}_C(C')$ into $\text{detcyc}(\mu)$ since $\text{skel}(\mu)$ contains C and C' as cycles.

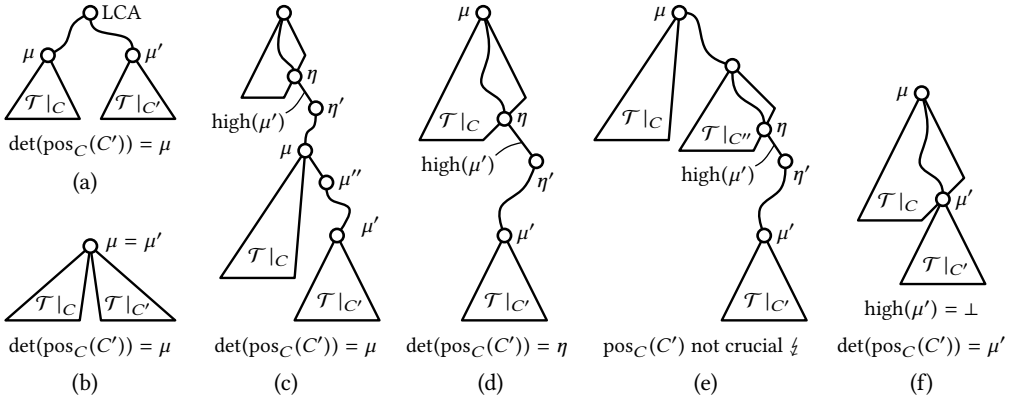


Figure 7.7: Illustration of the cases that occur in the proof of Lemma 7.8.

In the third case the node determining $\text{pos}_C(C')$ lies somewhere on the path from μ down to μ'. In this situation $\text{high}(\mu')$ comes into play and we distinguish several cases. We first assume that $\text{high}(\mu') \neq \bot$. Let $\{\eta, \eta'\} = \text{high}(\mu')$ be the highest edge in $\mathcal{T}$ on the path from μ' to the root that is reachable without using an edge in any of the induced trees, as computed by Lemma 7.7. Let η be the parent of η'. We claim that either μ or η determines the crucial relative position $\text{pos}_C(C')$.

More precisely, if η lies above or is equal to μ (Figure 7.7c), then the child μ'' of μ on the path from μ' to μ does not contain C as a cycle. Otherwise the edge $\{\mu, \mu''\}$ would have been contained in $\mathcal{T}|_C$, which is a contradiction to the definition of $\text{high}(\mu')$. Thus, C' is contracted in the virtual edge ε in $\text{skel}(\mu)$ corresponding to the child μ'' that is not contained in the cycle induced by C, implying that μ determines $\text{pos}_C(C')$. In this case we set $\det(\text{pos}_C(C')) = \mu$. Moreover, we want to insert the crucial relative position $\text{pos}_C(C')$ into $\text{contr}(\varepsilon)$. Unfortunately, we cannot determine the virtual edge ε belonging to the child μ'' in constant time. We handle that problem by storing a temporary list $\text{temp}(\mu)$ for the node μ and insert $\text{pos}_C(C')$ into this list. After we have processed all crucial relative positions, we process $\mathcal{T}$ bottom-up, building a union-find data structure by taking the union of μ with all its children after processing μ. Thus, when processing μ, we can simply traverse the list $\text{temp}(\mu)$ once, find for every crucial relative position $\text{pos}_C(C')$ the virtual edge ε containing C' by finding $\text{root}(\mathcal{T}|_{C'})$ in the union-find data structure and then add $\text{pos}_C(C')$ to $\text{contr}(\varepsilon)$. Note that this takes overall linear time, because the union-find data structure consumes amortized constant time per operation since the union-operations we apply are known in advance [GT85].

In the second case η lies below μ, where $\text{high}(\mu') = \{\eta, \eta'\}$. We claim that η contains C as a cycle and C' contracted in the virtual edge ε' in $\text{skel}(\eta)$ corresponding to the child η', as depicted in Figure 7.7d. By definition of $\text{high}(\mu')$ there is a cycle C'' that is contained as a cycle in η and in the parent of η but not in η'. We show that $C'' = C$ or $\text{pos}_C(C')$ is not a crucial relative position. Assume $C'' \neq C$; see Figure 7.7e.

In skel(η) the cycle C' is contracted in the virtual edge ε' corresponding to the child η', whereas C is contracted in the virtual edge ε corresponding to the parent of η. Since C'' is a cycle in η and in its parent, it induces a cycle in skel(η) containing ε. Consider a path π from C' to C in the graph G. Then π contains one of the poles of skel(η) and hence contains a vertex in C''. Thus the relative position $\text{pos}_C(C')$ is not crucial, which is a contradiction. Hence we can simply set $\det(\text{pos}_C(C')) = \eta$ and append $\text{pos}_C(C')$ to the list contr(ε').

Finally, high(μ') may be not defined, i.e., high(μ') = $\perp$ since the edge connecting μ' to its parent is already contained in one of the induced cycle trees. With a similar argument as before, this induced tree is $\mathcal{T}|_C$, belonging to the cycle C, as depicted in Figure 7.7f. Thus μ' contains C and C' as cycles and we set $\det(\text{pos}_C(C')) = \mu'$ and add $\text{pos}_C(C')$ to the list detcyc(μ'). This concludes the proof. □

In the fourth and last phase we process the SPQR-tree $\mathcal{T}$ once more to finally compute the PR-node constraints restricted to the crucial relative positions. We obtain the following lemma.

Lemma 7.9. *Let G be a biconnected planar graph. The PR-node constraints restricted to the crucial relative positions can be computed in linear time.*

Proof. We process each node in the SPQR-tree $\mathcal{T}$ of G once, consuming time linear in the size of its skeleton plus some additional costs that sum up to the number of crucial relative positions in total. Let μ be a node in $\mathcal{T}$. If μ is not contained in any induced tree $\mathcal{T}|_C$, it does not determine any relative position at all. Thus assume there is at least one cycle that is a cycle in μ. If μ is a P-node, skel(μ) consists of ℓ parallel virtual edges $\varepsilon_1, \dots, \varepsilon_\ell$ and we can assume without loss of generality that the cycle C induces in skel(μ) the cycle κ consisting of the two virtual edges ε_1 and ε_2. For every crucial relative position $\text{pos}_C(C')$ that is determined by μ there is a virtual edge $\varepsilon \in \{\varepsilon_3, \dots, \varepsilon_\ell\}$ containing C' in the list contr(ε), which is already computed due to Lemma 7.8. Hence, the PR-node constraints stemming from μ can be computed by processing each of these lists contr(ε), setting $\text{pos}_C(C') = \text{pos}_C(C'')$ for any two cycles C' and C'' appearing consecutively in contr(ε). The time-consumption is linear in the size of skel(μ) plus the number of crucial relative positions determined by μ.

If μ is an R-node, it may contain several cycles as a cycle, all of them stored in the list cyc(μ) due to Lemma 7.6. Every crucial relative position $\text{pos}_C(C')$ determined by μ is either contained in the lists contr(ε) for a virtual edge ε in skel(μ) or in detcyc(μ) if C and C' are both cycles in μ (Lemma 7.8). We first carry the relative positions in detcyc(μ) over to the corresponding cycles. More precisely, we define a list detcyc(C') for every cycle C' in cyc(μ) and insert a crucial relative position $\text{pos}_C(C')$ into it, if it is contained in detcyc(μ). This can obviously be done consuming time linear in the size of detcyc(μ). Afterwards, we start by fixing the embedding of skel(μ) and pick an arbitrary vertex v_0 in skel(μ). For each cycle C contained as cycle κ in skel(μ) we

define a variable side(C) and initialize it with the value "left" or "right", depending on which side v_0 lies with respect to κ in the chosen embedding of skel(μ), or with the value "on" if v_0 is contained in C. Due to Lemma 7.6 we know for every edge ε to which cycle it belongs (or that it does not belong to a cycle at all). Thus side(C) can be easily computed for every cycle C that is a cycle in skel(μ) consuming time linear in $|\operatorname{skel}(\mu)|$ by traversing skel(μ) once, starting at v_0.

To sum up, each crucial relative position $\operatorname{pos}_C(C')$ determined by μ is either contained in contr(ε) if C' is contracted in ε or in detcyc(C') if C' is a cycle in skel(μ). Moreover, for each cycle C the value of side(C) describes on which side of C the chosen start-vertex v_0 lies with respect to a chosen orientation of skel(μ). We now want to divide the crucial relative positions determined by μ into two lists Left and Right depending on which value they have with respect to the chosen embedding. If this is done, the PR-node constraints stemming from μ restricted to the crucial relative positions can be computed by simply processing these two lists once. To construct the lists Left and Right, we make a DFS-traversal in skel(μ) such that each virtual edge is processed once. More precisely, when we visit an edge $\{u, v\}$ (starting at u), then v is either an unvisited vertex and we continue the traversal from v or v was already visited, then we go back to u. If all virtual edges incident to the current vertex u were already visited, we do a back-tracking step, i.e., we go back to the vertex from which we moved to u. Essentially, a normal step consists of three phases, leaving the current vertex u, traveling along the virtual edge $\{u, v\}$, and finally arriving at v or back at u. In a back-tracking step we have only two phases, namely leaving the current vertex u and arriving at its predecessor. During the whole traversal we keep track of the sides side($\cdot$). More precisely, when leaving a vertex u that was contained in a cycle C we may have to update side(C) if the target-vertex v is not also contained in C. On the other hand, when arriving at a vertex v contained in a cycle C we have to set side(C) = "on". Since such an update has to be done for at most one cycle we can keep track of the sides in constant time per operation and thus in overall linear time. Now it is easy to compute the values of the crucial relative positions determined by μ with respect to the currently chosen embedding. While traveling along a virtual edge $\varepsilon = \{u, v\}$ we process contr(ε). For a crucial relative position $\operatorname{pos}_C(C')$ contained in contr(ε) we know that C' is contracted in ε. Thus, in the chosen embedding the value of $\operatorname{pos}_C(C')$ is the current value of side(C) and we can insert $\operatorname{pos}_C(C')$ into the list Left or Right depending on the value of side(C). This takes linear time in the number of crucial relative positions contained in contr(ε). We deal with the crucial relative positions contained in one of the lists detcyc(C') in a similar way. Every time we reach a vertex v contained in a cycle C' we check whether this is the first time we visit the cycle C'. If it is the first time, we insert every crucial relative positions $\operatorname{pos}_C(C')$ contained in detcyc(C') into one of the lists Left or Right, depending on the current value of side(C). Clearly the whole traversal takes linear time in the size

of skel(μ) plus linear time in the number of crucial relative positions determined by μ. Moreover, we obviously obtain the PR-node constraints restricted to the crucial relative positions stemming form μ by processing each of the lists Left and Right once, obtaining an equality constraint for positions that are adjacent in the lists and additionally a single inequality for a pair of positions, one contained in Left and the other in Right, unless one of them is empty. □

Corollary 7.1. *The CC-tree $\mathcal{T}_C$ of a biconnected planar graph G can be computed in linear time.*

It remains to extend the described algorithm to the case where G is not necessarily biconnected. More precisely, we need to show how to compute the extended PR-node constraints and the cutvertex constraints in linear time. This is done in the proof of the following theorem.

Theorem 7.6. *The CC-tree $\mathcal{T}_C$ of a connected planar graph G can be computed in linear time.*

Proof. As before, the underlying C-tree can be easily computed in linear time. For a fixed block B we have the SPQR-tree $\mathcal{T}$ and for a cycle C in B the induced tree $\mathcal{T}|_C$ can be defined as before. Obviously, Lemma 7.6 can be used as before to compute cyc(μ) for every node μ, bel(ε) for every virtual edge and root($\mathcal{T}|_C$) for every induced subtree in linear time. Moreover, the edge high(μ) in $\mathcal{T}$ can be computed for every node μ as in Lemma 7.7. For the computation of det(pos$_C(C')$) for every crucial relative position pos$_C(C')$ and contr(ε) for every virtual edge ε, we cannot directly apply Lemma 7.8 since the cycles C and C' may be contained in different blocks. Thus, before we can compute det(pos$_C(C')$), we need to find out whether C and C' are in the same block, which can be done by simply storing for every cycle a pointer to the block containing it. For the case that C and C' are contained in the same block det(pos$_C(C')$) can be computed as before and pos$_C(C')$ can be inserted into the list contr(ε) for some ε if necessary. For the case that C and C' are contained in different blocks B and B', we need to find the unique cutvertex v in B that separates B and B'. This can be done in overall linear time by computing the BC-tree and using an approach combining the lowest common ancestor and union-find data structure similar as in the proof of Lemma 7.8.

If the resulting cutvertex v is not contained in C, we can treat the cutvertex v as if it was the cycle C' and use the same algorithm as in Lemma 7.8 to compute det(pos$_C(C')$) and append pos$_C(C')$ to contr(ε) for some ε if necessary. If v is contained in C, then the crucial relative position pos$_C(C')$ is not determined by any node in any SPQR-tree at all, but by the embedding of the blocks around the cutvertex v. Thus there are no extended PR-node constraints restricting pos$_C(C')$. Finally, det(pos$_C(C')$) can be computed in overall linear time for every crucial relative position pos$_C(C')$ that is

determined by a node in the SPQR-tree of the block containing C. Moreover, for a node μ in the SPQR-tree of the block B containing C every virtual edge ε has a list $\text{contr}(\varepsilon)$ containing all crucial relative positions $\text{pos}_C(C')$ that are determined by μ and for which either C' is contracted in ε or belongs to a different block B' and is connected to B via a cutvertex contained in the expansion graph $\text{expan}(\varepsilon)$. With these information the extended PR-node constraints can be computed exactly the same as the PR-node constraints are computed in Lemma 7.9.

It remains to compute the cutvertex constraints restricted to the crucial relative positions. As mentioned above, we can compute a list of crucial relative positions $\text{pos}_C(C_1), \ldots, \text{pos}_C(C_\ell)$ determined by the embedding of the blocks around a cutvertex v contained in C in linear time. We then process this list once, starting with $\text{pos}_C(C_1)$. We store $\text{pos}_C(C_1)$ as reference position for the block B_1 containing C_1. Now, when processing $\text{pos}_C(C_i)$, we check whether the block B_i containing C_i already has a reference position $\text{pos}_C(C_j)$ assigned to it. In this case we set $\text{pos}_C(C_i) = \text{pos}_C(C_j)$, otherwise we set $\text{pos}_C(C_i)$ to be the reference position. This obviously consumes overall linear time and computes the cutvertex constraints restricted to the crucial relative positions. □

Intersecting CC-Trees in Linear Time

Due to Theorem 7.5 we can test whether two graphs $G^①$ and $G^②$ with common graph C consisting of a set of disjoint cycles have a simultaneous embedding by computing the CC-trees $\mathcal{T}_C^①$ and $\mathcal{T}_C^②$ of $G^①$ and $G^②$, respectively, which can be done in linear time due to Theorem 7.6. Then the intersection $\mathcal{T}_C$ of $\mathcal{T}_C^①$ and $\mathcal{T}_C^②$ represents exactly the possible embeddings of the common graph G in a simultaneous embedding. It remains to show that the intersection can be computed in linear time.

Theorem 7.7. *The intersection of two CC-trees can be computed in linear time.*

Proof. Let $\mathcal{T}_C^①$ and $\mathcal{T}_C^②$ be two CC-trees on a set $\mathcal{C}$ of cycles. We start with $\mathcal{T}_C = \mathcal{T}_C^①$ and show how to compute the common-face and crucial-position constraints in overall linear time. For the crucial-position constraints we essentially only show how to find for each crucial relative position in $\mathcal{T}_C^②$ a crucial relative position in $\mathcal{T}_C$ corresponding to it. Computing the crucial-position constraints is then easy. We root $\mathcal{T}_C$ at an arbitrary vertex and again use that the lowest common ancestor of two vertices in $\mathcal{T}_C$ can be computed in constant time [HT84; BF00]. For every edge $e^② = \{C_1, C_2\}$ in $\mathcal{T}_C^②$ we obtain a path in $\mathcal{T}_C$ from C_1 to the lowest common ancestor of C_1 and C_2 and further to C_2. We essentially process these two parts of the path separately with some additional computation for the lowest common ancestor. We say that the parts of the paths *belong* to the *half-edges* $e_1^②$ and $e_2^②$, respectively. We use the following data structure. For every cycle C there is a list $\text{end}(C)$ containing all edges in $\mathcal{T}_C^②$ whose endpoints have C in $\mathcal{T}_C$ as lowest common ancestor. This list can be computed

for every cycle in overall linear time. We then process $\mathcal{T}_C$ bottom up, saving for the cycle C we currently process a second list $\text{curr}(C)$ containing all the half-edges in $\mathcal{T}_C^{②}$ whose paths contain C. This can be done in overall linear time by ensuring that every half-edge $e_i^{②}$ $(i = 1, 2)$ is contained in at most one list $\text{curr}(C)$ at the same time. Then $e_i^{②}$ can be removed from this list in constant time by storing for $e_i^{②}$ pointers to the previous and to the next element in that list, denoted by $\text{prev}(e_i^{②})$ and $\text{next}(e_i^{②})$. Additionally, we build up the following union-find data structure. Every time we have processed a cycle C, we union C with all its children in $\mathcal{T}_C$. Thus, when processing C, this data structure can be used to find for every cycle in the subtree below C the child of C it belongs to. Note that again this version of the union-find data structure consumes amortized constant time per operation since the sequence of union operations is known in advance [GT85]. Before starting to process $\mathcal{T}_C$, we process $\mathcal{T}_C^{②}$ once and for every edge $e^{②} = \{C_1, C_2\}$ we insert the half-edges $e_1^{②}$ and $e_2^{②}$ to the lists $\text{curr}(C_1)$ and $\text{curr}(C_2)$, respectively. While processing $\mathcal{T}_C$ bottom up the following invariants hold at the moment we start to process C.

1. The list $\text{curr}(C)$ contains all half-edges starting at C.
2. For every child C' of C the list $\text{curr}(C')$ contains the half-edge $e_i^{②}$ if and only if the path belonging to it contains C and C'.
3. Every half-edge $e_i^{②}$ is contained in at most one list $\text{curr}(C)$, and $\text{prev}(e_i^{②})$ and $\text{next}(e_i^{②})$ contain the previous and next element in that list, respectively.

When we start processing a leaf C the invariants are obviously true. To satisfy invariant 2. for the parent of C we have to ensure that all half-edges ending at C are removed form the list $\text{curr}(C)$. Since there are no half-edges ending in a leaf, we simply do nothing. Invariants 1. and 3. obviously also hold for the parent of C.

Let C be an arbitrary cycle and assume that the invariants are satisfied. To ensure that invariant 2. holds for the parent of C, we need to build a list of all half-edges whose paths contain C and do not end at C. Since invariants 1. and 2. hold for C this are exactly the half-edges contained in $\text{curr}(C)$ plus the half-edges contained in $\text{curr}(C')$ for each of the children C' of C that are not ending at C. Note that a half-edge may also start at C and end at C. This is the case if the corresponding edge connects C with another cycle C'' such that the lowest common ancestor of C and C'' is C. We first process the list $\text{end}(C)$ containing the edges ending at C; let $e^{②}$ be an edge in $\text{end}(C)$. The two half-edges $e_1^{②}$ and $e_2^{②}$ belonging to $e^{②}$ are contained in the lists $\text{curr}(C_1)$ and $\text{curr}(C_2)$, where C_1 and C_2 are different cycles and each of them is either C or a child of C. We remove $e_i^{②}$ form $\text{curr}(C_i)$ for $i = 1, 2$. This can be done by setting the pointers $\text{next}(\text{prev}(e_i^{②})) = \text{next}(e_i^{②})$ and $\text{prev}(\text{next}(e_i^{②})) = \text{prev}(e_i^{②})$, taking constant time per edge since each half-edge is contained in at most one list due to invariant 3. Afterwards, for every child C' of C, we append $\text{curr}(C')$ to $\text{curr}(C)$ and

empty the list curr(C') afterwards, to ensure that invariant 3 remains satisfied. This takes constant time per child and thus overall time linear in the degree of C. Obviously this satisfies all invariants for the parent of C. Furthermore, we consume time linear in the degree of C plus time linear in the number of half-edges ending at C. However, every half-edge ends exactly once yielding overall linear time.

Now it is easy to compute the common-face and crucial-position constraints while processing $\mathcal{T}_C$ as described above. Essentially, when processing C, we compute all the constraints concerning the relative position of other cycles with respect to C. In particular, we need to add common-face constraints if two half-edges end at C and if the path belonging to a half-edge contains C in its interior. Furthermore, we find a corresponding crucial relative position for every half-edge starting at C. Let $e^{②} = \{C_1, C_2\}$ be an edge whose half-edges end at C. There are two different cases. First, one of the cycles C_i (for $i = 1, 2$) is C (its half edge starts and ends at C). Then the other cycle (whose half-edge only ends at C) is contained in a subtree with root C', where C' is a child of C. Second, C_1 and C_2 are contained in the subtrees with roots C_1' and C_2', respectively, where C_1' and C_2' are different children of C. In this case, both half-edges end at C. We consider the second case first. Then C_1' and C_2' can be found in amortized constant time by finding C_1 and C_2 in the union-find data structure. The equation $\text{pos}_C(C_1') = \text{pos}_C(C_2')$ is exactly the common-face constraint at the cycle C stemming from the edge $e^{②}$. In the second case we can again find the child C' in constant time. Assume without loss of generality that $C_1 = C$ and C_2 is contained in the subtree having C' as root. Then $\text{pos}_C(C')$ is the crucial relative position in $\mathcal{T}_C$ corresponding to the crucial relative position $\text{pos}_C(C_2)$ in $\mathcal{T}_C^{②}$. The half-edges containing C in its interior are exactly the half-edges contained in one of the lists curr(C') for a child C' of C whose path does not end at C. Thus, for the parent C'' of C we have to add the common-face constraint $\text{pos}_C(C') = \text{pos}_C(C'')$ if and only if the list curr(C') is not empty after deleting all half-edges in end(C). These additional computations obviously do not increase the running time and hence the common-face and crucial-position constraints can be computed in overall linear running time. □

Theorems 7.4, 7.5, 7.6 and 7.7 directly yield the following results.

Theorem 7.8. *SEFE can be solved in linear time if the common graph consists of disjoint cycles.*

Theorem 7.9. *SEFE can be solved in linear time for the case of k graphs $G^{①}, \ldots, G^{ⓚ}$ all intersecting in the same common graph G consisting of disjoint cycles.*

7.4 Connected Components with Fixed Embedding

In this section we show how the previous results can be extended to the case that the common graph has several connected components, each of them with a fixed planar

embedding. Again, we first consider the case of a single graph G containing $\mathcal{C}$ as a subgraph, where in this case $\mathcal{C}$ is a set of connected components instead of a set of disjoint cycles. First note that the relative position $\text{pos}_C(C')$ of a component C' with respect to another component C can be an arbitrary face of C. Thus, the choice of the relative positions is no longer binary and a set of inequalities on the relative positions would lead to a coloring problem in the conflict graph, which is $\mathcal{NP}$-hard in general. However, most of the constraints between relative positions are equations, in fact, all inequalities stem from R-nodes in the SPQR-tree of G (or of the SPQR-tree of one of the blocks in G). Fortunately, if a relative position is determined by an R-node, there are only two possibilities to embed this R-node. Thus, the possible values for the relative position is restricted to two faces, yielding a binary decision. Note that in general the possible values for $\text{pos}_C(C')$ are not all faces of C, even if $\text{pos}_C(C')$ is not determined by an R-node but by a P-node or by the embedding around a cutvertex.

Thus, we obtain for each relative position a set of possible faces as values and additionally several equations and inequalities, where inequalities only occur between relative positions with a binary choice. These conditions can be modeled as a conflict graph where each node represents a relative position with some allowed colors (faces) and edges in this conflict graph enforce both endvertices to be either colored the same or differently. In the case of the problem Sefe each of the graphs yields such a conflict graph. These conflict graphs can be easily merged by intersecting for each relative position the sets of allowed colors (faces). Then a simultaneous embedding can be constructed by first iteratively contracting edges requiring equality, intersecting the possible colors of the involved nodes. If the resulting graph contains a node with the empty set as choice for the color, then no simultaneous embedding exists. Otherwise, we have to test whether each connected component in the remaining graph can be colored such that adjacent nodes have different colors, which can be done efficiently since such a component either consists of a single node or there are only up to two possible colors for each connected component left due to the considerations above.

Moreover, the CC-tree can be adapted to work for the case of connected components with fixed embeddings instead of disjoint cycles, as the extended PR-node and cutvertex constraints on the crucial relative positions are still sufficient to imply them on all relative positions. We call this tree on connected components the CC$^{\oplus}$-tree, standing for *constrained component-tree*. In the following we quickly go through the steps we did before in the case of disjoint cycles and describe the changes when considering connected components instead.

PR-Node Constraints. Let G be a biconnected planar graph and let $\mathcal{C}$ be a subgraph of G consisting of several connected components, each with a fixed planar embedding. Let further $C \in \mathcal{C}$ be one of the connected components and let μ be a node in the SPQR-tree $\mathcal{T}$ of G. The virtual edges in $\text{skel}(\mu)$ whose expansion graphs contain parts

of the component C induce a connected subgraph in skel(μ). In the previous case, where the subgraph consisted of disjoint cycles, this induced subgraph was either a single edge or a cycle. In the case that C is an arbitrary component the induced graph can be an arbitrary connected subgraph of skel(μ). If it is a single edge, we say that C is *contracted* in μ, otherwise C is a *component* in μ.

We obviously obtain that the relative position $\text{pos}_C(C')$ of another component C' with respect to C is determined by the embedding of skel(μ) if and only if C is a component in μ and C' is not contracted in one of the virtual edges belonging to the subgraph induced by C. Moreover, the embedding of skel(μ) is partially (or completely) fixed by the embedding of C if the induced graph in skel(μ) contains a vertex with degree greater than 2. More precisely, consider μ to be a P-node containing C as a component. Then the virtual edges belonging to C have a fixed planar embedding and each face in this induced graph represents a face in C. These faces are the possible values for the relative positions with respect to C that are determined by μ. The remaining virtual edges not belonging to C can be added arbitrarily and thus components contracted in these edges can be put into one of the possible faces with the restriction that two components contracted in the same virtual edge have to lie in the same face, i.e., they have the same relative position with respect to C. To sum up, we obtain a set of possible faces of C with respect to μ and a set of equations between relative positions of components with respect to C.

For the case that μ is an R-node, either the embedding of skel(μ) is fixed due to the fact that there exists a component whose induced graph in skel(μ) contains a vertex with degree greater than 2. Otherwise, each component is either contracted in μ or the induced subgraph is a cycle or a path. No matter which case arises, the relative positions determined by μ are either completely fixed or there are only two possibilities. If the embedding is fixed, the relative positions determined by μ are fixed and thus there is no need for additional constraints. Otherwise, a crucial relative position with respect to C is fixed if C induces a path in skel(μ) and it changes by flipping skel(μ) if C induces a cycle. For two components C and C' both inducing a cycle in skel(μ) this yields a bijection between the two possible values for relative positions with respect to C determined by μ and the two possible values for positions with respect to C'. Thus we can add the equations and inequalities as in the case of disjoint cycles.

The resulting constraints are again called PR-node constraints. As for disjoint cycles we obtain that an embedding of the components C respecting the fixed embeddings for each component can be induced by an embedding chosen for G if and only if the PR-node constraints are satisfied. This directly yields a polynomial-time algorithm to solve Sefe for the case that both graphs are biconnected and the common graph consists of several connected components, each having a fixed planar embedding.

Extended PR-Node and Cutvertex Constraints. As for cycles the considerations above can be easily extended to the case that the graph G containing the components $\mathcal{C}$ is allowed to contain cutvertices. For a cutvertex v not contained in a component C, the relative position of v with respect to C determines the relative positions of components attached via v, which again yields the extended PR-node constraints. If v is contained in C, then the relative position of another component C' with respect to C is determined by the embedding around v if and only if v splits C from C'. In this case C' can obviously lie in one of the faces of C incident to v. Fortunately, the cutvertex constraints do not contain inequalities as they only ensure that components attached to v via the same block lie in the same face of C. With these considerations all results form Section 7.3.1 can be extended to the case of components with fixed embedding instead of cycles. In particular, Sefe can be solved in polynomial time if the common graph consists of connected components, each with a fixed planar embedding.

CC$^{\oplus}$-Trees. As mentioned before, the CC-tree can be adapted to represent all embeddings that can be induced on the set of components $\mathcal{C}$ by an embedding of the graph G, yielding the CC$^{\oplus}$-tree. To this end, each node in the tree represents a component $C \in \mathcal{C}$ and the incidence to C of an edge $\{C, C'\}$ in the CC$^{\oplus}$-tree represents the choice for the crucial relative position $\text{pos}_C(C')$. The possible values are restricted to a subset of faces of C as described before and there may be some equations between crucial relative positions with respect to C. Moreover, there may be inequalities between crucial relative positions even with respect to different components. However, if this is the case, then there are at most two possible choices and we have a bijection between the possible faces of different components. As in the proof of Theorem 7.4, it follows from the structure of the underlying C-tree, that relative positions that are not crucial are determined by a crucial relative position that is determined by the same P- or R-node or by the same embedding choice around a cutvertex. The proof can be easily adapted to the case of components instead of cycles yielding that satisfying the constraints and restrictions to a subset of faces for the crucial relative positions automatically satisfies these conditions for all relative positions.

To be able to solve Sefe with the help of CC$^{\oplus}$-trees, we need to intersect two CC$^{\oplus}$-trees such that the result is again a CC$^{\oplus}$-tree. Assume as in the case of cycles that we have the two CC$^{\oplus}$-trees $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$. As before we start with $\mathcal{T}_C^{①}$ and add the restrictions given by $\mathcal{T}_C^{②}$. More precisely, for every pair $\{C, C'\}$ of adjacent nodes in $\mathcal{T}_C^{②}$ we have to add the common-face constraints to $\mathcal{T}_C^{①}$, i.e., equations between crucial relative positions on the path between C and C' in $\mathcal{T}_C^{①}$ enforcing C and C' to share a face. Moreover, for every relative position $\text{pos}_C(C')$ that is crucial with respect to $\mathcal{T}_C^{②}$ we have to add the equations and inequalities it is involved in to the CC$^{\oplus}$-tree $\mathcal{T}_C^{①}$. As for cycles $\text{pos}_C(C')$ is in $\mathcal{T}_C^{①}$ determined by the crucial relative position $\text{pos}_C(C'')$, where C'' is the first node on the path from C to C'. We have to

do two things. First, we have to restrict the possible choices for $\text{pos}_C(C'')$ to those that are possible for $\text{pos}_C(C')$, which can easily be done by intersecting the two sets. Second, the equations and inequalities $\text{pos}_C(C')$ is involved in have to be carried over to $\text{pos}_C(C'')$. This can be done as before by choosing for each crucial relative position in $\mathcal{T}_C^{②}$ the representative in $\mathcal{T}_C^{①}$. For the resulting intersection $\mathcal{T}_C$ it remains to show that every embedding represented by it is also represented by $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$. The former is clear, the latter can be shown as in the proof of Theorem 7.5.

Efficient Implementation. Unfortunately, the constrained component-tree may have quadratic size in contrast to the constrained cycle-tree, which has linear size. This comes from the fact that a node C in the CC$^{\oplus}$-tree may have linearly many neighbors. Moreover, each relative position $\text{pos}_C(C')$ of a neighbor C' of C may have linearly many possible values, as C may have that many faces. As these possible values need to be stored for the edge $\{C, C'\}$ in the CC-tree it has quadratic size. On the other hand, it is easy to see that the CC-tree can be computed in quadratic time. Moreover, the proof of Theorem 7.7 providing a linear-time algorithm to intersect CC$^{\oplus}$-trees can be adapted almost literally. The only thing that changes is that additionally the possible values for $\text{pos}_C(C'')$ and $\text{pos}_C(C')$ need to be intersected, where $\text{pos}_C(C')$ is a relative position that is crucial with respect to $\mathcal{T}_C^{②}$ and $\text{pos}_C(C'')$ is the representative in $\mathcal{T}_C^{①}$. Thus, two CC$^{\oplus}$-trees can be intersected consuming time linear in the size of the CC$^{\oplus}$-trees, i.e., quadratic time in the size of the input graphs. We finally obtain the following theorem.

Theorem 7.10. *SEFE can be solved in quadratic time, if the embedding of each connected component of the common graph is fixed.*

Let $\mathcal{T}_C$ be the CC$^{\oplus}$-tree representing all embeddings of the components C that can be induced by the graph G. It is worth noting that, although the explicit representation of $\mathcal{T}_C$ may have quadratic size, it also admits a compact representation of linear size. The key idea is the following. In case there are more than two possible values for a crucial relative position $\text{pos}_C(C')$, this position is determined by a P-node or a cutvertex. Then we can encode the possible values for $\text{pos}_C(C')$ by pointing to a list that is stored at that P-node or cutvertex, respectively. Since this set of values is independent of C' it is sufficient to store one list for each P-node or cutvertex. It is not hard to see that the total size of these lists is linear. Moreover, the fast algorithm for computing CC-trees can be applied with obvious modifications to compute this compact representation in linear time. It is, however, unclear whether the intersection of two or more CC$^{\oplus}$-trees still admits a compact representation and whether it can be computed in linear time from the given compact representations.

7.5 Conclusion

Contrary to the previous results on simultaneous embeddings, we focused on the case where the embedding choice does not consist of ordering edges around vertices but of placing connected components in relative positions to one another. We first showed that both input graphs of an instance of Sefe can be always assumed to be connected. We then showed how to solve Sefe in linear time for the case that the common graph consists of simple disjoint cycles (or more generally has maximum degree 2). We further extended the result to a quadratic-time algorithm solving the more general case where the embedding of each connected component of the common graph is fixed. These solutions include a compact and easy to handle data structure, the CC-tree and $CC^{\oplus}$-tree, representing all possible simultaneous embeddings. We will see in Chapter 8 that the techniques developed in this chapter can actually be used and extended to also work in more general cases.

8 Edge Orderings, Relative Positions, and Cutvertices in Simultaneous Planarity

In this chapter, we combine the techniques from Chapter 7 ensuring consistent relative positions with existing and newly developed approaches for ensuring consistent edge orderings. We present two types of results. First, a set of three linear-time preprocessing algorithms that remove certain substructures from a given Sefe instance, producing a set of equivalent Sefe instances without such substructures. The structures we can remove are (1) cutvertices of the union graph, (2) most separating pairs of the union graph, and (3) connected components of the common graph that are biconnected but not a cycle.

Second, we give an $O(n^3)$-time algorithm solving Sefe for instances with the following restriction. Let u be a pole of a P-node μ in the SPQR-tree of $G^{①}$ or $G^{②}$. Then at most three virtual edges of μ may contain common edges incident to u. All algorithms extend to the sunflower case, i.e., to the case of more than three graphs pairwise intersecting in the same common graph.

This chapter is based on joint work with Annette Karrer and Ignaz Rutter [BKR13a].

8.1 Introduction

There are two fundamental approaches to solving Sefe in the literature. The first approach is based on the characterization of Jünger and Schulz [JS09] stating that finding a simultaneous embedding of two graphs $G^{①}$ and $G^{②}$ with common graph G is equivalent to finding planar embeddings of $G^{①}$ and $G^{②}$ that induce the same embedding on G. The second very recent approach by Schaefer [Sch13] is based on Hanani-Tutte-style redrawing results. One tries to characterize the existence of a Sefe via the existence of drawings of the union graph $G^{\cup}$ where no two independent edges of the same graph cross an odd number of times. The existence of such drawings can be expressed using a linear system of boolean equations.

When following the first approach, we need two things to describe the planar embedding of the common graph G. First, for each vertex v, a cyclic order of incident edges around v. Second, for every pair of connected components H and H' of G, the face f of H containing H', i.e., the relative position of H' with respect to H. To find a simultaneous embedding, one needs to find a pair of planar embeddings that induce the same cyclic edge orderings (consistent edge orderings) and the same relative positions (consistent relative positions) on the common graph G.

Most previous results use the first approach but none of them considers both, consistent edge orderings and relative positions. Most of them assume the common graph to be connected or to contain no cycles, making it sufficient to ensure consistent relative positions. The strongest results of this type are the two linear-time algorithms for the case that G is biconnected by Haeupler et al. [HJL13] and by Angelini et al. [Ang+12] and a quadratic-time algorithm for the case where $G^{①}$ and $G^{②}$ are biconnected and G is connected [BR13]. In the latter result, Sefe is modeled as an instance of the problem Simultaneous PQ-Ordering. On the other hand, as shown in the previous chapter, there is a linear-time algorithm for Sefe if the common graph consists of disjoint cycles, which requires to ensure consistent relative positions but makes edge orderings trivially consistent.

The advantage of the second approach (Hanani-Tutte) is that it implicitly handles both, consistent edge orderings and consistent relative positions, at the same time. Thus, the results by Schaefer [Sch13] are the first that handle Sefe instances where the common graph consists of several, non-trivial connected components. He gives a polynomial-time algorithm for the cases where each connected component of the common graph is biconnected or has maximum degree 3. Although this approach is conceptionally simple, very elegant, and combines several notions of planarity within a common framework, it has two disadvantages. The running time of the algorithms are quite high and the high level of abstraction makes it difficult to generalize the results.

Contribution and Outline

In this chapter, we follow the first approach and show how to enforce consistent edge orderings and consistent relative positions at the same time, by combining different recent approaches, namely the algorithm by Angelini et al. [Ang+12], the result on Simultaneous PQ-Ordering [BR13] (see also Chapter 5) for consistent edge orderings, and the result on disjoint cycles (Chapter 7) for consistent relative positions. Note that the relative positions of connected components to each other are usually expressed in terms of faces (containing the respective component). This is no longer possible if the embeddings, and thus the set of faces, of connected components are not fixed. To overcome this issue, we show that these relative positions can be expressed in terms of relative positions with respect to cycles in a cycle basis. In addition to that, we are able to handle certain cutvertices of $G^{①}$ and $G^{②}$.

We classify a vertex v to be a *union cutvertex*, a *simultaneous cutvertex*, and an *exclusive cutvertex* if v is a cutvertex of $G^{\cup}$, of $G^{①}$ and $G^{②}$ but not of $G^{\cup}$, and of $G^{①}$ but not $G^{②}$ or the other way around, respectively. Similarly, we can define *union separating pairs* to be separating pairs in $G^{\cup}$. We present several preprocessing algorithms that simplify given instances of Sefe; see Section 8.2. Besides a very technical preprocessing step (Section 8.2.4), they remove union cutvertices and most (but not all) union

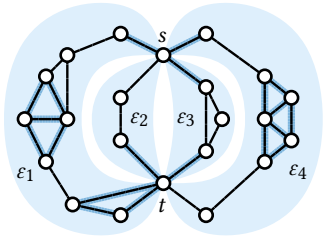


Figure 8.1: A P-node of $G^{①}$ with virtual edges $\varepsilon_1, \ldots, \varepsilon_4$. The node has common P-node degree 3; for s the virtual edges ε_1, ε_3, and ε_4 count; for t the virtual edges ε_1, ε_2, and ε_3 count.

separating pairs; see Theorem 8.3, and replace connected components of G that are biconnected with cycles. They run in linear time and can be applied independently. The latter algorithm together with the linear-time algorithm for disjoint cycles (Theorem 7.8 in Chapter 7) improves the result by Schaefer [Sch13] for instances where every connected component of G is biconnected to linear time.

In Section 8.4 we show how to solve instances that have common P-node degree 3 and simultaneous cutvertices of common degree at most 3 in cubic time. A vertex has *common degree* k if it is a common vertex with degree k in G. To define common P-node degree, let μ be a P-node of $G^{①}$. We say that μ has *common P-node degree* k if both vertices in skel(μ) are incident to common edges in the expansion graphs of at most k virtual edges (note that these can be different edges for the two vertices); see Figure 8.1 for an example. We say that an instance of Sefe has common P-node degree k if the P-nodes of $G^{①}$ and the P-nodes of $G^{②}$ all have common P-node degree k. Our algorithm for instances with common P-node degree 3 and simultaneous cutvertices of common degree 3 relies heavily on the preprocessing algorithms that exclude certain structures. Together with the preprocessing steps, our algorithm can in particular solve a given Sefe instance if every connected component of G is biconnected, has maximum degree 3, or is outerplanar with maximum degree 3 cutvertices. As before, this also applies to the sunflower case.

8.2 Preprocessing Algorithms

In this section, we present several algorithms that can be used as a preprocessing of a given Sefe instance. The result is usually a set of Sefe instances that admit a solution if and only if the original instance admits one. The running time of the preprocessing algorithms is linear, and so is the total size of the equivalent set of Sefe instances. Each of the preprocessing algorithms removes certain types of structures form the instance, in particular from the common graph. Namely, we show that we can eliminate union cutvertices, simultaneous cutvertices with common-degree 3, and connected components of G that are biconnected but not a cycle. None of these algorithms introduces new cutvertices in G or increases the degree of a vertex. Thus,

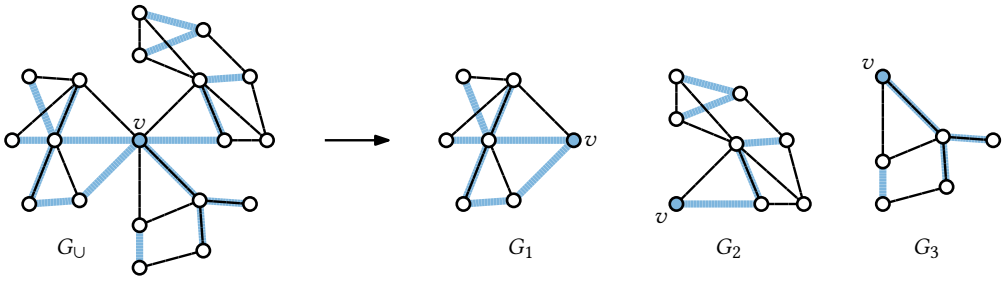


Figure 8.2: A union cutvertex separates a SEFE instance into independent subinstances.

the preprocessing algorithms can be successively applied to a given instance, removing all the claimed structures.

Let $(G^①, G^②)$ be a SEFE instance with common graph $G = G^① \cap G^②$. We can equivalently encode such an instance in terms of its *union graph* $G^\cup = G^① \cup G^②$, whose edges are labeled $\{1\}$, $\{2\}$, or $\{1,2\}$, depending on whether they are contained exclusively in $G^①$, exclusively in $G^②$, or in G, respectively. Any graph with such an edge coloring can be considered as a SEFE instance. Since sometimes the coloring version is more convenient, we use these notions interchangeably throughout this section.

8.2.1 Union Cutvertices

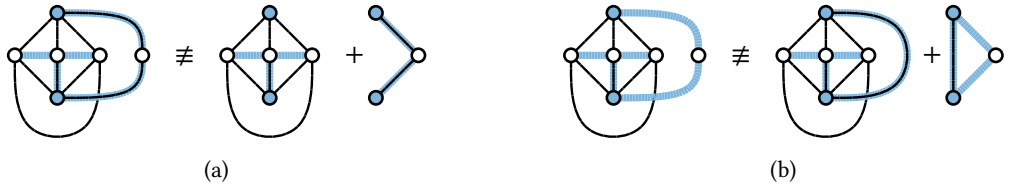
Recall that a union cutvertex of a SEFE instance $(G^①, G^②)$ is a cutvertex of the union graph $G^\cup$. The following theorem states that the SEFE instances corresponding to the split components of a cutvertex of $G^\cup$ can be solved independently; see Figure 8.2.

Lemma 8.1. *Let $G^\cup$ be a SEFE instance and let v be a cutvertex of $G^\cup$ with split components $G_1, \ldots, G_k$. Then $G^\cup$ admits a SEFE if and only if G_i admits a SEFE for $i = 1, \ldots, k$.*

Proof. Clearly, a SEFE of $G^\cup$ contains a SEFE of $G_1, \ldots, G_k$. Conversely, given a SEFE $\mathcal{E}_i$ of G_i for $i = 1, \ldots, k$, we can assume without loss of generality that v is incident to the outer face in each of the $\mathcal{E}_i$. Then these embeddings can be merged to a SEFE $\mathcal{E}$ of $G^\cup$. □

Due to Lemma 8.1, it suffices to consider the blocks of $G^\cup$ of a SEFE instance independently. Clearly, the blocks can be computed in $O(n)$ time, and, given a SEFE for each block, a SEFE of the original instance can be computed in $O(n)$ time.

Theorem 8.1. *There is a linear-time algorithm that decomposes a SEFE instance into an equivalent set of SEFE instances that do not contain union cutvertices.*



(a) (b)

Figure 8.3: (a) The original instance does not admit a SEFE but the split components with respect to the separating pair $\{u, v\}$ (marked vertices) do. (b) The original instance admits a SEFE but one of the split components does not when adding the common edge uv.

8.2.2 Union Separating Pairs

In analogy to a union cutvertex, we can define a *union separating pair* to be a separating pair of the union graph $G^{\cup}$. It is tempting to proceed as for the union cutvertices: separate $G^{\cup}$ according to a union separating pair, solve the subinstances corresponding to the resulting subgraphs, and merge the partial solutions.

However, this approach fails as merging the partial solutions may be impossible; see Figure 8.3a. Note that it is easy to merge the partial solutions if all of them have u and v on the outer face of their union graph. One can enforce this kind of behaviour by connecting u and v with a common edge in each subinstance. Unfortunately, this is too restrictive as the subinstances may fail to have a SEFE with this additional edge whereas the original instance has a solution; see Figure 8.3b.

We can, however, use the idea of adding the common edge uv in every subinstance to get rid of most union separating pairs. Throughout the whole section, we assume that u and v are vertices of the same block B of the common graph and that $\{u, v\}$ is a separating pair in B. If $\{u, v\}$ separates B into three or more split components, then u and v are poles of a P-node of $\mathcal{T}(B)$. The case when there are only two split components is a somewhat special (less interesting) case. To achieve a more concise notation, we thus assume in the following that u and v are the poles of a P-node. However, all arguments extend to the special case with two split components.

Let μ be the P-node of $\mathcal{T}(B)$ with poles u and v. Two virtual edges ε_1 and ε_2 of skel(μ) are *linked in* $G^{①}$ if $G^{①}$ contains a path from an inner vertex in expan(ε_1) to an inner vertex in expan(ε_2) that is disjoint from B (except for the endvertices of the path). The *①-link graph* $L_\mu^{①}$ of μ has the virtual edges of μ as nodes, with an edge between two nodes if and only if the corresponding virtual edges are linked in $G^{①}$. Analogously, we can define the *②-link graph* $L_\mu^{②}$ and the *union-link graph* $L_\mu^{\cup}$.

Note that the $L_\mu^{①}$ and $L_\mu^{②}$ are subgraphs of $L_\mu^{\cup}$. But $L_\mu^{\cup}$ is no the union of $L_\mu^{①}$ and $L_\mu^{②}$, as two virtual edges may be linked in the union graph but in none of the two exclusive graphs; see Figure 8.4. However, the union of $L_\mu^{①}$ and $L_\mu^{②}$ will also be of interest later. We call it the *exclusive-link graph* and denote it by $L_\mu^{①} \cup L_\mu^{②}$. An edge in

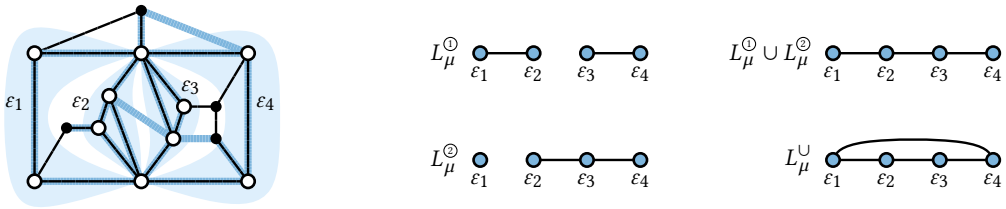


Figure 8.4: A P-node μ of the union graph with four virtual edges $\varepsilon_1, \ldots, \varepsilon_4$ together with the link graphs $L_\mu^{①}$, $L_\mu^{②}$, $L_\mu^{①} \cup L_\mu^{②}$, and $L_\mu^{\cup}$.

the exclusive-link graph indicates that the two corresponding virtual edges are either linked in $G^{①}$ or in $G^{②}$.

We note that the following two lemmas are neither entirely new (e.g., Angelini et al. use a slightly weaker statement [Ang+12]) nor very surprising.

Lemma 8.2. *Let $L_\mu^{\cup}$ be a union-link graph of a given* Sefe *instance and let ε_1 and ε_2 be adjacent in $L_\mu^{\cup}$. In every simultaneous embedding, ε_1 and ε_2 are adjacent in the embedding of* skel(μ).

Proof. First assume that ε_1 and ε_2 are already adjacent in $L_\mu^{①}$. Then the expansion graphs of ε_1 and ε_2 bound a face in every embedding of G that extends to an embedding of $G^{①}$. Thus, ε_1 and ε_2 must be adjacent in the embedding of skel(μ). The same holds if ε_1 and ε_2 are adjacent in $G^{②}$.

Otherwise, let B be the block whose SPQR-tree contains μ. Let π be a path in the union graph connecting inner vertices v_1 and v_2 in the expansion graphs of ε_1 and ε_2, respectively, that is disjoint from B. Clearly, then common vertices of π must be embedded into a face B that is incident to v_1 and to v_2. Such a face only exists if ε_1 and ε_2 are adjacent in the embedding of skel(μ). □

Lemma 8.3. *If $(G^{①}, G^{②})$ admits a* Sefe*, then each union-link graph is either a cycle or a collection of paths.*

Proof. Let B be a block of G and let μ be a P-node of $\mathcal{T}(B)$. Let skel(μ) be embedded according to a simultaneous embedding of $(G^{①}, G^{②})$. Let $\varepsilon_1, \ldots, \varepsilon_k$ be the virtual edges of skel(μ) embedded in this order. Due to Lemma 8.2, two virtual edges ε_i and ε_j can be adjacent in $L_\mu^{\cup}$ only if $i + 1 = j$ or $i = k$ and $j = 1$. Thus, $L_\mu^{\cup}$ is a subgraph of the cycle $\varepsilon_1, \ldots, \varepsilon_k, \varepsilon_1$. Hence, $L_\mu^{\cup}$ is either a cycle or a collection of paths. □

Assume the union-link graph $L_\mu^{\cup}$ of a P-node μ is connected (i.e., by Lemma 8.3 a cycle or a path containing all virtual edges). Then Lemma 8.2 implies that the virtual edges in skel(μ) have to be embedded in a fixed order up to reversal. In this case, it remains to choose between two different embeddings, although the k virtual edges

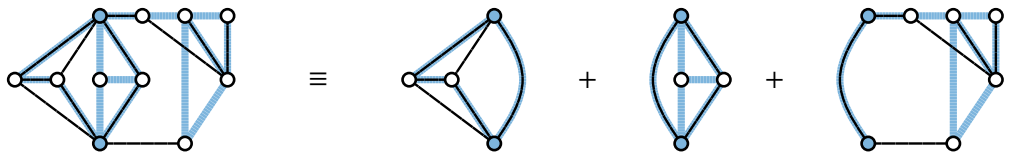


Figure 8.5: A union separating pair that separates a common cycle can be used to decompose the instances into simpler parts.

of skel(μ) have $(k-1)!$ different cyclic orders. In the following we show that we can assume without loss of generality that every union-link graph is connected.

Assume $L_\mu^\cup$ is not connected. Then the poles u and v of μ are a separating pair in the union graph. Moreover, the expansion graphs of two virtual edges from different connected components of $L_\mu^\cup$ end up in different split components with respect to u and v. Thus, we get at least two split components with a common path from u to v. If this is the case, we say that the separating pair $\{u,v\}$ *separates a common cycle*. We obtain the following lemma; see Figure 8.5.

Lemma 8.4. *Let $\{u,v\}$ be a separating pair of the union graph $G^\cup$ that separates a common cycle and let $G_1^\cup,\dots,G_k^\cup$ be the split components. Then $G^\cup$ admits a* Sefe *if and only if $G_i^\cup$ with the additional common edge uv admits a* Sefe *for $i=1,\dots,k$.*

Proof. Assume we have a solution for each subinstances $G_i^\cup + uv$. As uv is a common edge, we can assume without loss of generality that it lies in the boundary of the outer face. It is thus easy to obtain a drawing of $G^\cup$ from these partial solutions without introducing any new crossings. Thus, this yields a Sefe of $G^\cup$.

Conversely, assume $G^\cup$ admits a Sefe. As $\{u,v\}$ separates a common cycle, we can assume that G_1 and G_2 both contain a path of common edges connecting u and v. We have to show that $G_i + uv$ admits a Sefe for every $i=1,\dots,k$. Assume that $i \neq 1$. Let π be the path of common edges connecting u and v in G_1. The graph $G_i + \pi$ (which is a subgraph of $G^\cup$) admits a Sefe as the property of admitting a Sefe is closed under taking subgraphs. Moreover, it is also closed under contracting common edges. Thus, we can assume that π is actually the common edge uv. This yields a Sefe of $G_i + uv$. For $i=1$ we can use the common path connecting u and v in G_2 instead. □

As argued above, a disconnected union-link graph implies the existence of a separating pair that separates a common cycle. We thus obtain the following theorem.

Theorem 8.2. *There is a linear-time algorithm that decomposes a* Sefe *instance into an equivalent set of* Sefe *instances of total linear size in which all union-link graphs are connected.*

Proof. Clearly, applying the decomposition implied by Lemma 8.4 exhaustively results in a set of instances of total linear size. It remains to show that we can apply all

decomposition steps in total linear time. To this end, consider the SPQR-tree $\mathcal{T}$ of the union graph $G^\cup$. Note that $G^\cup$ is non-planar in general and thus the R-nodes skeletons of $\mathcal{T}$ may be non-planar. Nonetheless, $\mathcal{T}$ can be computed in linear time [GM01] and represents all separating pairs of $G^\cup$.

Let μ be an inner node of $\mathcal{T}$ and let $\varepsilon = uv$ be a virtual edge in skel(μ). We say that ε is a *common virtual edge* if the expansion graph of ε includes a common uv-path from. Note that $\{u,v\}$ is a separating pair of $G^\cup$. Moreover, if we know for each virtual edge whether it is a common virtual edge, we can determine whether $\{u,v\}$ separates a common cycle by only looking at skel(μ). More precisely, if μ is a P-node, then $\{u,v\}$ separates a common cycle if and only if two or more virtual edges are common virtual edges. For S- and R-nodes, $\{u,v\}$ separates a common cycle if and only if the virtual edge ε is a common virtual edge and skel(μ) $-\ \varepsilon$ includes a path of common virtual edges from u to v.

Let us assume, we know for each virtual edge, whether it is a common virtual edge. Then we can easily compute the decomposition by rooting $\mathcal{T}$ and processing it bottom up. Thus, it remains to compute the common virtual edges in linear time. To this end, first root $\mathcal{T}$ at a Q-node. By processing $\mathcal{T}$ bottom up, one can easily compute for each virtual edge, except for the parent edges, whether it is a common virtual edge or not.

It remains to deal with the parent edges. We process $\mathcal{T}$ top down. When processing a node μ, we assume that we know the common virtual edges of skel(μ) (potentially including the parent edge). We then compute in $O(|\operatorname{skel}(\mu)|)$ time for which children of μ, the parent edge is a common virtual edge. If μ is the root (i.e., a Q-node), then the only child of μ has a common virtual edge as parent edge if and only if the edge corresponding to the Q-node μ is a common edge.

Let μ be a P-node and let ε be a virtual edge in skel(μ). Then twin(ε) (which is the parent edge of the child corresponding to ε) is a common virtual edge if and only if skel(μ) includes a common virtual edge different from ε. Thus, μ can be processed in $O(|\operatorname{skel}(\mu)|)$ time. If μ is an S-node, it similarly holds that twin(ε) is a common virtual edge if and only if all virtual edges of skel(μ) except maybe ε are common virtual edges.

Finally, if μ is an R-node, consider the graph skel$'(\mu)$ obtained from skel(μ) by deleting all non-common virtual edges. Let ε be an arbitrary virtual edge of skel(μ). If ε is non-common, then twin(ε) is a common virtual edge if and only if the endvertices of ε lie in the same connected component of skel$'(\mu)$. If ε is a common virtual edge, then twin(ε) is a common virtual edge if and only if ε is not a bridge in skel$'(\mu)$. Note that both of these properties can be checked in constant time for each virtual edge of skel(μ) after $O(|\operatorname{skel}(\mu)|)$ preprocessing time. Thus, we can also process R-nodes in $O(|\operatorname{skel}(\mu)|)$ time, which yields an overall linear running time. □

Let B be a block of the common graph and let μ be a P-node of $\mathcal{T}(B)$. By Theorem 8.2, we can assume that the union-link graph $L_\mu^\cup$ is connected. Thus, the ordering of the

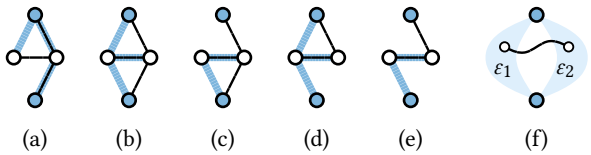


Figure 8.6: (a–e) Common, exclusive, ①-, ②-, and union connected split components (in this order). (f) The face between two virtual edges that are ①-linked.

virtual edges in skel(μ) is fixed up to reversal. Hence, the embedding choices for μ are the same as those for an R-node.

In the following, we provide further simplifications by eliminating some types of simultaneous separating pairs. Let u and v be the poles of the P-node μ. Consider the case that $\{u,v\}$ is a separating pair in the union graph $G^\cup$ with split components $G_1^\cup,\dots,G_k^\cup$ (we can assume by Theorem 8.1 that neither u nor v is a cutvertex in $G^\cup$). As before, we denote the common graph and the exclusive graphs corresponding to the SEFE instances $G_i^\cup$ (for $i=1,\dots,k$) by G_i, $G_i^①$ and $G_i^②$, respectively.

We define $G_i^\cup$ to be *common connected* if u and v are connected by a path in G_i; see Figure 8.6a. The split component $G_i^\cup$ is *exclusive connected*, if it is not common connected but u and v are connected by exclusive paths in both graphs $G_i^①$ and $G_i^②$; see Figure 8.6b. It is ①*-connected*, if u and v are connected by a path in $G_i^①$ but not in $G_i^②$; see Figure 8.6c. The term ②*-connected* is defined analogously; see Figure 8.6d. Note that being ①- or ②-connected excludes being common or exclusive connected. Finally, if $G_i^\cup$ is neither of the above, it is *union connected*; see Figure 8.6e.

We say that μ is an *impossible P-node* if $L_\mu^①$ is a cycle and one of the split components is ①-connected, if $L_\mu^②$ is a cycle and one of the split components is ②-connected, or if $L_\mu^① \cup L_\mu^②$ is a cycle and one of the split components is exclusive connected.

Lemma 8.5. *A SEFE instance with an impossible P-node is a no-instance.*

Proof. Let $G_1^\cup,\dots,G_k^\cup$ be the split components with respect to the poles u and v of the P-node μ. As μ is an impossible P-node, the union-link graph $L_\mu^\cup$ is a cycle. Thus, at most one split component can be common connected. As u and v are the poles of a P-node of the common graph, one of the split components must be common connected. Thus, exactly one split component, without loss of generality $G_1^\cup$, is common connected.

First assume that μ is an impossible P-node due to the fact that $L_\mu^①$ is a cycle and one of the split components, without loss of generality $G_2^\cup$ is ①-connected.

Assume the given SEFE instance $G^\cup$ admits a SEFE and assume $G_1^\cup$ and $G_2^\cup$ are embedded according to this SEFE. As $G_2^\cup$ is ①-connected (Figure 8.6c), the graph $G_2^①$ includes a path $\pi^①$ from u to v. Clearly, $\pi^①$ lies in a single face of $G_1^①$. Let f be the corresponding face of the common graph G_1. The boundary of f belongs to the expansion graphs of two different virtual edges ε_1 and ε_2 of skel(μ); see Figure 8.6f. However, ε_1 and ε_2 cannot be ①-linked (as in Figure 8.6f), as otherwise $\pi^①$ could not be

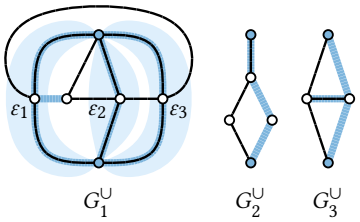


Figure 8.7: The union split component $G_1^\cup$ includes the expansion graphs of all three virtual edges ε_1, ε_2, and ε_3. The edge pairs $\varepsilon_1, \varepsilon_3$ and $\varepsilon_2, \varepsilon_3$ are ①-linked; thus the exclusive connected split component $G_2^\cup$ cannot be embedded into the faces between ε_1 and ε_3 or between ε_2 and ε_3. Although ε_1 and ε_2 are neither ①- nor ②-linked, $G_2^\cup$ cannot be embedded into the face between ε_1 and ε_2 due to its common end. The component $G_3^\cup$ has no common end and can be embedded into the face between ε_1 and ε_2.

embedded into the face f without having a crossing in $G^①$. It follows that $L_\mu^①$ cannot be a cycle, a contradiction.

Analogously, if $L_\mu^②$ is a cycle and one of the split components is ②-connected, we find a path $\pi^②$ that is a witness for a pair of adjacent virtual edges that are not ②-linked. It remains to consider the case where $L_\mu^① \cup L_\mu^②$ is a cycle and $G_2^\cup$ is exclusive connected. In this case, $G_2^①$ and $G_2^②$ include paths $\pi^①$ and $\pi^②$, respectively, connecting u and v. As they both belong to the same split component, they have to be embedded in the same common face of G_1. Thus, there are adjacent virtual edges that are neither ①- nor ②-linked. Hence, $L_\mu^① \cup L_\mu^②$ is not a circle. □

Due to this lemma, it is sufficient to consider the case that μ is not an impossible P-node. We want to show that the different split components (in the union graph, with respect to the poles u and v of μ) can be handled independently. However, we have to exclude a special case to make this true. Let $G_i^\cup$ be one of the split components that is exclusive connected. We say that $G_i^\cup$ has *common ends* if it contains a common edge incident to u or to v. Figure 8.7 shows an example, where the following lemma does not hold without excluding exclusive connected components with common ends.

Lemma 8.6. *Let $G^\cup$ be a* Sefe *instance and let μ be a non-impossible P-node whose poles are a separating pair with split components $G_1^\cup, \ldots, G_k^\cup$. Assume $G_1^\cup$ is the only common connected split component and none of the exclusive connected components has common ends. Then $G^\cup$ admits a* Sefe *if and only if $G_1^\cup$ admits a* Sefe *and $G_i^\cup$ together with the common edge uv admits a* Sefe *for $i = 2, \ldots, k$.*

Proof. Assume that $G_1^\cup$ and $G_i^\cup + uv$ (for $i = 2, \ldots, k$) admit simultaneous embeddings. We show how to combine the simultaneous embeddings of $G_1^\cup$ and $G_2^\cup + uv$ to a simultaneous embedding of $G_1^\cup \cup G_2^\cup$. The procedure can then be iteratively applied to the other split components. We have to distinguish the cases that $G_2^\cup$ is union connected, ①-connected, ②-connected, and exclusive connected (without common ends).

First assume that $G_2^\cup$ is union connected. Figure 8.8 shows an example illustrating the proof for this case. As u and v are the poles of a P-node, the common graph G_1 has

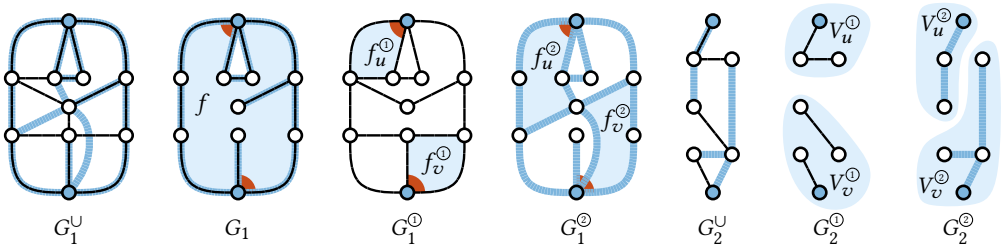


Figure 8.8: Two split components of the union graph illustrating the proof of Lemma 8.6.

a face f that is incident to u and to v. Let further $f_u^{①}$ and $f_v^{①}$ be faces of $G_1^{①}$ incident to u and v, respectively, that are both part of the union face f. Similarly, we choose faces $f_u^{②}$ and $f_v^{②}$ in $G_2^{②}$ that are incident to u and v, respectively, and that are both part of f. Note that u might have several incidences to the face f, i.e., when u is a cutvertex in G_1 and one of the corresponding blocks is embedded into f. In this case, we choose $f_u^{②}$ such that it has *the same incidence* to u as $f_u^{①}$, i.e., the common edges appearing in the cyclic order around u before and after $f_u^{②}$ are the same as those that appear before and after $f_u^{②}$. We ensure the same for $f_v^{①}$ and $f_v^{②}$. In the example in Figure 8.8, f has two incidences to u and two incidences to v and the chosen incidence is marked by an angle.

Due to the common edge uv, we can assume that the SEFE of $G_2^{\cup}$ has uv and uv on the outer face. As $G_2^{\cup}$ is not ①-connected, we can separate the vertices of $G_2^{①}$ into two subsets $V_u^{①}$ and $V_v^{①}$, such that $V_u^{①}$ contains all vertices of the connected component of $G_2^{①}$ containing u, while $V_v^{①}$ contains all other vertices. We can then embed the vertices of $V_u^{①}$ into $f_u^{①}$ and the vertices of $V_v^{①}$ into $f_v^{①}$ without changing the embedding of $G_2^{①}$. In the same way, $G_2^{②}$ can be embedded into $f_u^{②}$ and $f_v^{②}$.

As we did not change the embedding of $G_1^{\cup}$ or $G_2^{\cup}$, the edge orderings are consistent for all vertices except maybe u and v. Moreover, the relative positions between connected components in $G_1^{\cup}$ is consistent and the same holds for $G_2^{\cup}$. As the four faces $f_u^{①}$, $f_v^{①}$, $f_u^{②}$, and $f_v^{②}$ belong to the same common face f, the relative positions of components in G_2 with respect to components in G_1 are also consistent. Moreover, all components of G_1 lie in the outer face of G_2 with respect to $G_2^{①}$ and $G_2^{②}$. Finally, the edge ordering at u is consistent, as all edges incident to u in $G_2^{①}$ and $G_2^{②}$ are embedded between the same pair of common edges in G_1. As the same holds for v, we obtain a simultaneous embedding of $G_1^{\cup} \cup G_2^{\cup}$.

If $G_2^{\cup}$ is ①-connected, we know that $L_\mu^{①}$ is not a circle (otherwise, μ would be impossible). Thus, we can choose the faces f, $f_u^{①}$, and $f_v^{①}$ such that $f_u^{①} = f_v^{①}$. Then we can embed $G_2^{①}$ into this face without separating it. All remaining arguments work the same as above. The case that $G_2^{\cup}$ is ①-connected is symmetric.

Finally, if $G_2^{\cup}$ is exclusive connected, there is a pair of virtual edges that are neither ①- nor ②-linked. Thus, we can choose the common face f and the faces $f_u^{①}$, $f_v^{①}$, $f_u^{②}$,

and $f_v^{②}$ belonging to f such that $f_u^{①} = f_v^{①} = f^{①}$ and $f_u^{②} = f_v^{②} = f^{②}$. Unfortunately, we cannot always ensure that $f^{①}$ and $f^{②}$ have the same incidence to u or v; see Figure 8.7. However, the arguments form the previous cases still ensure that all relative positions and all cyclic orders except for maybe at u and v are consistent. As $G_2^{\cup}$ has no common ends, all common edges incident to u and v are contained in G_1 and thus the cyclic orders around these vertices are also consistent.

Note that combining the simultaneous embeddings $G_1^{\cup}$ and $G_2^{\cup}$ in this way (for all four cases) maintains the properties that there are faces in G_1, $G_1^{①}$, or $G_2^{②}$ that are incident to both poles u and v. Thus, we can continue adding all remaining subinstances $G_3^{\cup}, \ldots, G_k^{\cup}$ in the same way. □

Assume we exhaustively applied Lemma 8.4, Lemma 8.5, and Lemma 8.6 to a given instance of Sefe and let $(G^{①}, G^{②})$ be the resulting instance. Let u and v be the poles of a P-node μ of the common graph such that $\{u, v\}$ are a separating pair in the union graph. By Lemma 8.4 we can assume that $\{u, v\}$ does not separate a common cycle. Thus, exactly one split component has a common uv-path. By Lemma 8.5, we can assume that μ is a non-impossible P-node. Thus, we could apply Lemma 8.6 if there were split components without common ends. Hence we obtain the following theorem.

Theorem 8.3. *Let $(G^{①}, G^{②})$ be an instance of Sefe. In linear time, we can find equivalent instances such that every union separating pair $\{u, v\}$ has one of the following properties.*

- *The vertices u and v are not the poles of a P-node of a common block.*
- *Every split component has a common edge incident to u or to v but only one has a common uv-path.*

Proof. It remains to prove the claimed running time. The linear running time for decomposing the instances along its union separating pairs that separate a common cycle was already shown for Theorem 8.2. In Section 8.3.3 we extend the algorithm by Angelini et al. [Ang+12] for solving Sefe if the common graph is biconnected to the case where we allow exclusive vertices and have so-called union bridge constrains. It is not hard to see that testing $(G^{①}, G^{②})$ for the existence of impossible P-nodes can be done using the linear-time algorithm from Section 8.3.3.

It remains to decompose the union graph $G^{\cup}$ according to separating pairs that separated $G^{\cup}$ according to Lemma 8.6. As in the proof of Theorem 8.2, we consider the SPQR-tree $\mathcal{T}$ of $G^{\cup}$. For Theorem 8.2, we had to compute for every virtual edge, whether its expansion graph included a common path between its endpoints. Now, we in addition have to know which expansion graphs are exclusive connected and have common ends. This can be done analogously to the proof of Theorem 8.2. □

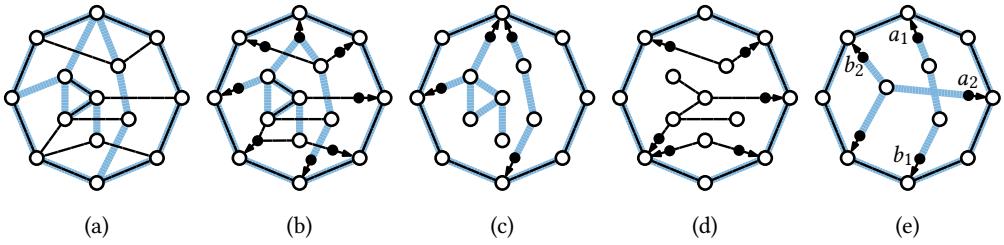


Figure 8.9: Situation where the connected component C of G is a cycle. (a) A simultaneous embedding of $(G^{①}, G^{②})$ with C on the outer face. (b) Removing C yields a single connected component in $G^{\cup}$. Thus, there is only one union bridge. Its attachment vertices are illustrated as black dots. (c) The two ①-bridges. (d) The three ②-bridges. Note that different bridges might share attachment vertices. (e) Two alternating ①-bridges.

8.2.3 Connected Components that are Biconnected

Let $(G^{①}, G^{②})$ be a SEFE instance and let C be a connected component of the common graph G that is a cycle; see Figure 8.9a. A *union bridge* of $G^{①}$ and $G^{②}$ with respect to C is a connected component of $G^{\cup} - C$ together with all its *attachment vertices* on C; see Figure 8.9b. Equivalently, the union bridges are the split components of $G^{\cup}$ with respect to the vertices of C excluding the edges of C. Similarly, there are ①*-bridges* and ②*-bridges*, which are connected components of $G^{①} - C$ and $G^{②} - C$ together with their attachment vertices on C, respectively; see Figure 8.9c–d. We say that two bridges B_1 and B_2 *alternate* if there are attachments a_1, b_1 of B_1 and attachments a_2, b_2 of B_2, such that the order along C is $a_1a_2b_1b_2$; see Figure 8.9e. We have the following lemma, which basically states that we can handle different union bridges independently

Lemma 8.7. *Let $G^{①}$ and $G^{②}$ be two planar graphs and let C be a connected component of the common graph that is a cycle. Then the graphs $G^{①}$ and $G^{②}$ admit a SEFE where C is the boundary of the outer face if and only if (i) each union bridge admits a SEFE together with C and (ii) no two i-bridges of C alternate for $i = 1, 2$.*

Proof. Clearly the conditions are necessary; we prove sufficiency. Let $B_1, \ldots, B_k$ be the union bridges with respect to C, and let $(\mathcal{E}_1^{①}, \mathcal{E}_1^{②}), \ldots, (\mathcal{E}_k^{①}, \mathcal{E}_k^{②})$ be the corresponding simultaneous embeddings of B_i together with C, which exist by condition (i). Note that each union bridge is connected, and hence all its edges and vertices are embedded on the same side of C. After possibly flipping some of the embeddings, we may assume that each of them has C with the same clockwise orientation as the outer face.

We now glue $\mathcal{E}_1^{①}, \ldots, \mathcal{E}_k^{①}$ to an embedding $\mathcal{E}^{①}$ of $G^{①}$, which is possible by condition (ii). In the same way, we find an embedding $\mathcal{E}^{②}$ of $G^{②}$ from $\mathcal{E}_1^{②}, \ldots, \mathcal{E}_k^{②}$. We claim that $(\mathcal{E}^{①}, \mathcal{E}^{②})$ is a SEFE of $G^{①}$ and $G^{②}$. For the consistent edge orderings, observe that any common vertex v with common-degree at least 3 is contained, together with all neighbors, in some union bridge B_i. The compatibility of the edge ordering follows

since $(\mathcal{E}_i^{①}, \mathcal{E}_i^{②})$ is a SEFE. Concerning the relative position of a vertex v and some common cycle C', we note that the relative positions clearly coincide in $\mathcal{E}^{①}$ and $\mathcal{E}^{②}$ for $C' = C$. Otherwise C' is contained in some union bridge. If v is embedded in the interior of C' in one of the two embeddings, then it is contained in the same union bridge as C', and the compatibility follows. If this case does not apply, it is embedded outside of C' in both embeddings, which is compatible as well. □

We note that this approach fails, when the cycle C is not a connected component of G, i.e., when a union bridge contains common edges incident to an attachment vertex. The reason is that the order of common edges incident to this attachment vertex is chosen in the moment one reinserts the union bridges into C.

Now consider a connected component C of the common graph G of a SEFE instance such that C is biconnected. Such a component is called *2-component*. If C is a cycle, it is a *trivial 2-component*. We define the union bridges, and the ①- and ②-bridges of $G^{①}$ and $G^{②}$ with respect to C as above. We call an embedding $\mathcal{E}$ of C together with an assignment of the union bridges to its faces *admissible* if and only if, (i) for each union bridge, all attachments are incident to the face to which it is assigned, and (ii) no two ①- and not two ②-bridges that are assigned to the same face alternate.

In the following, we try to solve the given SEFE instance $(G^{①}, G^{②})$ by first finding an admissible embedding of the 2-component C. Then we test for every face of C whether all union bridges can be embedded inside the corresponding facial cycle. By Lemma 8.7 we know that this is possible if and only if each union bridge together with the facial cycle admits a SEFE and no two ⓘ-bridges (for $i = 1, 2$) alternate. The latter is ensured by property (ii) of the admissible embedding of C. The former yields simpler SEFE instances in which the 2-component C is represented by a simple cycle. It remains to show that, if this approach fails, there exists no SEFE. First note that the properties (i) and (ii) of an admissible embedding are clearly necessary. Thus, if there is no admissible embedding of C, then there is no SEFE. It remains to show that it does not depend on the admissible embedding of C one chooses, whether a union bridge together with the facial cycle of the face it is assigned to admits a SEFE or not. In fact, the following lemma shows that the facial cycle one gets for a union bridge is more or less independent from the embedding of C, i.e., the attachment vertices of the bridge always appear in the same order along this cycle.

Lemma 8.8. *Let G be a biconnected planar graph and let X be a set of vertices that are incident to a common face in some planar embedding of G. Then the order of X in any simple cycle of G containing X is unique up to reversal.*

Proof. Consider a planar embedding $\mathcal{E}$ of G where all vertices in X share a face, and let C_X denote the corresponding facial cycle. Note that C_X is simple since G is biconnected. Let C be an arbitrary simple cycle in G containing all vertices in X. In $\mathcal{E}$, all parts of C that are disjoint from C_X are embedded outside of C_X. Let C'_X denote the

cycle obtained from C_X by contracting all maximal paths whose internal vertices do not belong to C to single edges. Observe that C_X and C'_X visit the vertices of X in the same order. Consider the graph $C \cup C'_X$, which is clearly outerplanar and biconnected. Hence both C and C'_X visit the vertices of X in the same order (up to complete reversal). Since C was chosen arbitrarily, the claim follows. □

For a union bridge B, let C_B denote the cycle consisting of the attachments of B in the ordering of an arbitrary cycle of G containing all the attachments. By Lemma 8.8, the cycle C_B is uniquely defined. Let further G_B denote the graph consisting of the union bridge B and the cycle C_B connecting the attachment vertices of B. We call this graph the *union bridge graph* of the bridge B. The following lemma formally states our above-mentioned strategy to decompose a Sefe instance.

Lemma 8.9. *Let $G^①$ and $G^②$ be two connected planar graphs and let C be a 2-component of the common graph G. Then the graphs $G^①$ and $G^②$ admit a Sefe if and only if (i) C admits an admissible embedding, and (ii) each union bridge graph admits a Sefe. If a Sefe exists, the embedding of C can be chosen as an arbitrary admissible embedding.*

Proof. Clearly, a Sefe of $G^①$ and $G^②$ defines an embedding of C and a bridge assignment that is admissible. Moreover, it induces a Sefe of each union bridge graph.

Conversely, assume that C admits an admissible embedding and each union bridge graph admits a Sefe. We obtain a Sefe of $G^①$ and $G^②$ as follows. Embed C with the admissible embedding and consider a face f of this embedding with facial cycle C_f. Let $B_1, \dots, B_k$ denote the union bridges that are assigned to this face, and let $(\mathcal{E}_1^①, \mathcal{E}_1^②), \dots, (\mathcal{E}_k^①, \mathcal{E}_k^②)$ be simultaneous embeddings of the bridge graphs G_B. By subdividing the cycle C_B, in each of the embeddings, we may assume that the outer face of each B_i in the embedding $(\mathcal{E}_i^①, \mathcal{E}_i^②)$ is the facial cycle C_f with the same orientation in each of them. By Lemma 8.7, we can hence combine them to a single Sefe of all union bridges whose outer face is the cycle C_f. We embed this Sefe into the face f of C. Since we can treat the different faces of C independently, applying this step for each face yields a Sefe of $G^①$ and $G^②$ with the claimed embedding of C. □

Lemma 8.9 suggests a simple strategy for reducing Sefe instances containing non-trivial 2-components. Namely, take such a component, construct the corresponding union bridge graphs, where C occurs only as a cycle, and find an admissible embedding of C. Finding an admissible embedding for C can be done as follows. To enforce the non-overlapping attachment property, replace each ①-bridge of C by a *dummy* ①-*bridge* that consists of a single vertex that is connected to the attachments of that bridge via edges in $E^①$. Similarly, we replace ②-bridges, which are connected to attachments via exclusive edges in $E^②$. We seek a Sefe of the resulting instance (where the common graph is biconnected), additionally requiring that dummy bridges belonging to the same union bridge are embedded in the same face. We also refer to such an instance as

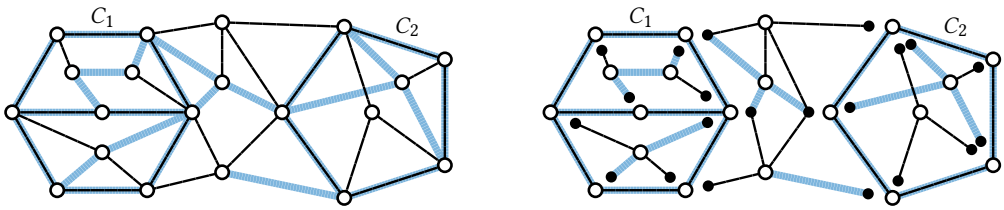


Figure 8.10: The instance on the left contains two 2-components C_1 and C_2. The corresponding union subbridges are shown on the right.

Sefe with union bridge constraints. A slight modification of the algorithm by Angelini et al. [Ang+12] can decide the existence of such an embedding in polynomial time. This gives the following lemma.

Lemma 8.10. *Computing an admissible embedding of a 2-component C is equivalent to solving Sefe with union bridge constraints on an instance having C as common graph. This can be done in polynomial time.*

It then remains to treat the union bridge graphs. Exhaustively applying Lemma 8.9 (using Lemma 8.10 to find admissible embeddings) results in a set of Sefe instances where each 2-component is trivial. Note that we could go even further and decompose along cycles that have more than one union bridge. However, this is not necessary to obtain the following theorem.

Theorem 8.4. *Given a Sefe instance, an equivalent set of instances of total linear size such that each 2-component of these instances is trivial can be computed in polynomial time.*

We can improve the running time in Theorem 8.4 to linear. However, it is quite tedious work, involving a lot of data structures, and results in a lengthy proof. To not disturb the reading flow too much, the proof is outsourced into its own section (Section 8.3, starting on page 218). Here, we only sketch it very roughly.

We do not apply an iterative process, removing one 2-component after another (as suggested above), but we decompose the whole instance at once. For this, we introduce the notion of *subbridges*. A subbridge of a graph G with respect to components $C_1, \dots, C_k$ is a maximal connected subgraph of G that does not become disconnected by removing all vertices of one component C_i; see Figure 8.10.

Recall that we have to deal with bridges in two ways. First, each 2-component forms a Sefe instance with its ⓘ-bridges, while the union bridges partition these ⓘ-bridges (yielding union bridge constraints). Second, each union bridge yields a union bridge graph, which is a simpler instance one has to solve. In both cases, one can deal with subbridges instead of the whole bridges for the following reason. For the first case, we need the ⓘ-bridges only to create the corresponding dummy bridges. Thus, it suffices

to know their attachment vertices. It is readily seen that each bridge B of C_i contains a unique subbridge S incident to C_i and that the attachments of S at C_i are exactly the attachments of B at C_i. As this also holds for the union bridges, the union subbridges already define the correct grouping of the ⓘ-subbridges. Concerning the second case, the Sefe instances that remain after exhaustively applying Lemma 8.9 are exactly the union subbridges together with a set of cycles, one for each incident 2-component. To conclude, it remains to show that each of the following three steps runs in linear time.

1. Compute for each 2-component the number of incident ①- and ②-subbridges, for each such subbridge its attachments, and the grouping of these subbridges into union subbridges.
2. Solve Sefe with union bridge constraints on instances with biconnected common graph.
3. Compute for each union subbridge a corresponding instance where each 2-component has been replaced by a suitable cycle.

For step 1 (Section 8.3.1), we contract every 2-component of $G^{\cup}$ into a single vertex. The union subbridges are then basically the split components with respect to the resulting vertices. The same holds for the ①- and ②-subbridges (using $G^{①}$ and $G^{②}$ instead of $G^{\cup}$). For step 2 (Section 8.3.3), we modify the algorithm due to Angelini et al. [Ang+12]. Augmenting it such that it computes admissible embeddings in polynomial time is straightforward. Achieving linear running time is quite technical and, like the linear version of the original algorithm, requires some intricate data structures. For Step 3 (Section 8.3.2), computing the union subbridges is easy. To compute a suitable cycle for each incident 2-component C_i, one can make use of the fact that we already know an admissible embedding of C_i from step 2.

Theorem 8.5. *Given a Sefe instance, an equivalent set of instances of total linear size such that each 2-component of these instances is trivial can be computed in linear time.*

8.2.4 Special Bridges and Common-Face Constraints

In Section 8.2.3, we considered the case that C is a 2-component of the common graph G. We called the split components with respect to the vertices of C bridges (excluding edges in C). Clearly, this definition extends to the case where C is an arbitrary connected component. However, the decomposition into smaller instances does not extend to this more general case as for example Lemma 8.8 fails for non-biconnected graphs. Nonetheless, we are able to eliminate some special types of bridges in exchange for so-called common-face constraints. The reduction we describe in this section is thus in a sense weaker than the previous reductions as we reduce a given Sefe instance to a set of equivalent instances with common-face constraints.

Let $(G^{①}, G^{②})$ be an instance of Sefe with common graph G and let $\mathcal{F} \subseteq 2^{V(G)}$ be a family of sets of common vertices. A given Sefe satisfies the *common-face constraints* $\mathcal{F}$ if and only if G has a face incident to all vertices in V' for every $V' \in \mathcal{F}$. The common-face constraints $\mathcal{F}$ are *block-local* if for every $V' \in \mathcal{F}$ all vertices in V' belong to the same block of G.

Similarly, we say union bridge B is *block-local* if all attachment vertices of B belong to the same block of C. Let $B_1^{ⓘ}, \ldots, B_k^{ⓘ}$ be the ⓘ-bridges (for $i \in \{1,2\}$) belonging to B. We say that B is *exclusively one-attached* if $B_j^{ⓘ}$ has only a single attachment vertex for $j = 1, \ldots, k$.

Let B be a block-local union bridge of the common connected component C. Then the attachment vertices of B appear in the same order in every cycle of C (Lemma 8.8). Thus, we can define the union bridge graph G_B of B as in Section 8.2.3. Consider the Sefe instance $(H^{①}, H^{②})$ obtained from $(G^{①}, G^{②})$ by removing the union bridge B (the attachment vertices are not removed). It follows from Section 8.2.3 that $(G^{①}, G^{②})$ admits a Sefe if and only if the union bridge graph G_B admits a Sefe, and $(H^{①}, H^{②})$ admits a Sefe with an assignment of B to one of its faces f such (i) all attachment vertices of B are incident to f, and (ii) for $i \in \{1,2\}$, no ⓘ-bridge in B alternates with another ⓘ-bridge in f.

If B is not only block-local but also exclusive one-attached, the latter requirement is trivially satisfied (a ⓘ-bridge that has only a single attachment vertex cannot alternate). Thus, it remains to test whether G_B admits a Sefe and $(H^{①}, H^{②})$ admits a Sefe with block-local common-face constraints. We obtain the following theorem. The linear running time can be shown as in Section 8.3.

Theorem 8.6. *Given a Sefe instance, an equivalent set of instances with block-local common-face constraints of total linear size can be computed in linear time such that each instances satisfies the following property. No union bridge of a common connected component that is not a cycle is block-local and exclusively one-attached.*

8.3 Preprocessing 2-Components in Linear Time

As promised in the end of Section 8.2.3, we prove in this section that the decomposition of a Sefe instance into equivalent instances where every 2-component is a cycle can be done in liner time. Readers who want to skip this section can continue with Section 8.4 in page 239.

8.3.1 Computing the Sefe-Instances with Union Bridge Constraints

We first consider a slightly more general setting. Let $G = (V, E)$ be a graph and let $C_1, \ldots, C_k$ be disjoint connected subgraphs of G. We are interested in computing the number of bridges of each connected component C_i together with the attachments

to C_i. We show that this can be done in $O(n + m)$ time (even if G is non-planar), where $n = |V|$ and $m = |E|$. Instead of computing directly the bridges and their attachments, our goal is rather to label each edge e that is incident to a vertex of some C_i but does not belong to any of the C_i itself, by the bridge of C_i that contains e. Observe that, if each such incident edge has been labeled, the information about the number of C_i-bridges and their attachments can easily be extracted by scanning all incidences of vertices of C_i for $i = 1, \ldots, k$. This scanning process can clearly be performed in total $O(n + m)$ time. In the following, we thus focus on computing this incidence labeling. Note that, since we are only interested in the attachments of bridges, it suffices to consider the corresponding subbridges as they have the same attachment sets.

Recall that a subbridge is a maximal connected subgraph of G for which none of the C_i is a separator. Note the high similarity of the definition of subbridges and the blocks of a graph, which are maximal connected subgraphs for which no single vertex is a separator. As with the blocks of a graph, it is readily seen that each edge of G that is not contained in one of the C_i is contained in exactly one subbridge of G. We exploit this similarity further and define the *component-subbridge tree* of G with respect to $C_1, \ldots, C_k$ as the graph that contains one vertex c_i for each component C_i and one vertex s_j for each subbridge S_j. Two vertices c_i and s_j are connected by an edge if and only if the subbridge S_j is incident to the component C_i. Note that, indeed, the component-subbridge tree is a tree. Once the component-subbridge tree has been computed, we can label each edge of $E \setminus (\bigcup_{i=1}^{k} C_i)$ with the subbridge containing it.

Lemma 8.11. *The component-subbridge tree of a graph G with respect to disjoint connected subgraphs $C_1, \ldots, C_k$ can be computed in linear time.*

Proof. First, contract each component C_i to a single vertex c_i; call the resulting graph G'. Note that, in G', the subbridges are exactly the maximal connected subgraphs for which none of the vertices c_i is a separator. We compute the component-subbridge tree T in three steps. First, compute the block-cutvertex tree of G'. Second, for each cutvertex v that does not correspond to one of the c_i, remove v and merge its incident blocks into the same subbridge. Finally, create for each component C_i that has only one bridge a corresponding vertex c_i and attach it as a leaf to the unique subbridge incident to C_i. Clearly, each of the steps can be performed in $O(n + m)$ time. □

As argued above, Lemma 8.11 can be used to label in linear time the incident edges of the components $C_1, \ldots, C_k$ by their corresponding bridges. For step 1 of our reduction, we take $C_1, \ldots, C_k$ as the 2-components of a SEFE instance $(G^{①}, G^{②})$. We then use the above approach to label the attachment incidences of the ①-, ②-, and the union bridges of $C_1, \ldots, C_k$. From this we can create the dummy bridges for each 2-component C_i together with the union bridge constraints in time linear in the sum of degrees of vertices in C_i. By the arguments for Lemma 8.10, the resulting instance admits a SEFE

if and only if C_i has an admissible embedding. Since the C_i are disjoint, it follows that the construction of all instances can be done in linear time. This finishes step 1.

8.3.2 Constructing the Subbridge Instances

Let us assume that each 2-component has an admissible embedding, which is found using the linear-time algorithm described in Section 8.3.3. Otherwise a Sefe of the original graph does not exist. In the final step of our reduction, we substitute, in each subbridge, the incident 2-components by a cycle. This results in a set of Sefe instances—one for each subbridge—that all admit a solution if and only if the original instance admits a solution. They can hence be handled completely independently. To efficiently extract all instances, we process the 2-components independently and replace each one by a cycle in their incident subbridges. The time to process a single 2-component with all its incident subbridges is linear in the size of the 2-component plus the number of attachments of these subbridges in the respective 2-component. It then immediately follows that processing all 2-components in this way takes linear time.

Consider a fixed 2-component C with an admissible embedding as computed in step 2 of the reduction. Consider a fixed face f together with the subbridges that are embedded in that face. For each bridge B embedded in f, we construct a list of attachments A_B. We traverse the facial cycle of f. At each vertex, we check the edges embedded inside this face and append the vertex to the list of each subbridge for which it is an attachment. Afterwards, we traverse for each subbridge its list of attachments and replace C by a cycle that visits the attachments in the order of the attachment list. The time is clearly proportional to the size of f and the attachments of the subbridges embedded in f. Hence processing all faces of all components in this way takes linear time and yields the claimed result. This implements step 3 in linear time.

8.3.3 Simultaneous Embedding with Union Bridge Constraints

In this section, we show how to solve Sefe with union bridge constraints in linear time if the common graph is biconnected. Our algorithm is based on the algorithm by Angelini et al. [Ang+12]. Note that our extension to this algorithm is twofold. We allow bridges with an arbitrary number of attachment vertices. The original algorithm allows only two attachments per ⓘ-bridge (i.e., each bridge is a single exclusive edge). Moreover, we have to deal with union bridge constraints. To avoid some special cases and simplify the description, we sometimes deviate from the notation used by Angelini et al. Our focus lies on a linear-time implementation, the correctness of our approach directly follows from the correctness of the algorithm by Angelini et al.

Let G be the biconnected common graph. We consider the (unrooted) *augmented SPQR-tree* $\mathcal{T}$ of G, which is defined as follows; see Figure 8.11. Let $\mathcal{T}'$ be the SPQR-tree

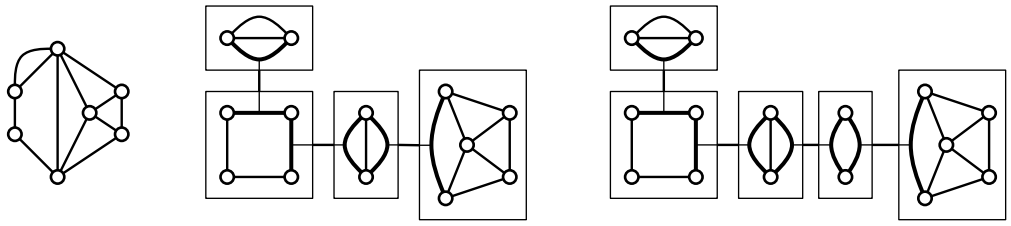


Figure 8.11: A graph (left) together with its SPQR-tree (middle). The Q-nodes are omitted to improve readability. The augmented SPQR-tree (right) contains an additional S-node whose skeleton is a cycle of length 2.

of G and let μ_1 and μ_2 be two adjacent nodes in $\mathcal{T}'$ such that each of them is a P- or an R-node. We basically insert a new S-node between μ_1 and μ_2 whose skeleton is a cycle of length 2 (i.e., a pair of parallel virtual edges). More precisely, let $\varepsilon_1 = \{s,t\}$ and $\varepsilon_2 = \{s,t\}$ be the virtual edges in μ_1 and μ_2, respectively, that correspond to each other. We subdivide the edge μ_1,μ_2 in $\mathcal{T}'$; let μ be the new subdivision vertex. The skeleton skel(μ) contains the vertices s and t with two virtual edges ε_1' and ε_2' between them corresponding to ε_1 in skel(μ_1) and ε_2 in skel(μ_2), respectively. Applying this augmentation for every pair of adjacent nodes that are P- or R-nodes gives the augmented SPQR-tree $\mathcal{T}$. Note that P- and R-nodes in $\mathcal{T}$ have only S- and Q-nodes as neighbors. Moreover, no two S-nodes are adjacent.

Consider a bridge B and let μ be a node of $\mathcal{T}$. A virtual edge ε in skel(μ) is an *attachment* of B if its expansion graph contains an attachment vertex of B. We say that B is *important* for μ if it has at least two distinct attachments among the vertices and virtual edges of skel(μ) that are not two adjacent vertices in skel(μ). It is clearly necessary, that skel(μ) admits an embedding such that for every union bridge B the attachments in skel(μ) are incident to a common face. An embedding having this property is called *compatible*. This leads to the following first step of the algorithm.

Step 1: Compatible embeddings. For every P- and R-node μ, compute the important union bridges with their attachments. If μ is an R-node, check whether the unique (up to flip) embedding skel(μ) is compatible. If μ is a P-node check whether it admits a compatible embedding and fix such an embedding (up to flip).

If Step 1 fails, the instance does not admit a Sefe. Note that the skeleton of a P-node might admit several compatible embeddings. However, fixing an arbitrary compatible embedding up to flip does not make a solvable Sefe instance unsolvable [Ang+12]. Thus, after Step 1, we can assume that the embedding of every skeleton is fixed and it remains to decide for each skeleton whether its embedding should be flipped or not. We call the embedding fixed for a skeleton its *reference embedding*.

For every P- and R-node μ let x_μ be a binary decision variable with the following interpretation. The skeleton skel(μ) is embedded according to its reference embedding

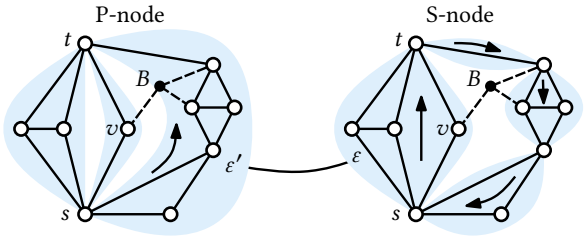


Figure 8.12: An S-node with a bridge B having ε as right-sided attachment. The virtual edges are illustrated as blue regions with their expansion graph inside.

and according to the flipped reference embedding if $x_\mu = 0$ and $x_\mu = 1$, respectively. By considering the S-nodes of the augmented SPQR-tree, one can derive necessary conditions for these variables that form an instance of 2-Sat (actually, we only get equations and inequalities, which is a special case of 2-Sat).

Let μ be an S-node. We assume the edges in skel(μ) to be oriented such that skel(μ) is a directed cycle. Thus, we can use the terms left face and right face to distinguish the faces of skel(μ). Let B be a bridge that is important for μ. We can either embed B into the left or into the right face of skel(μ). We define the binary variable x_B^μ with the interpretation that B is embedded into the right face and into the left face of skel(μ) if $x_B^\mu = 0$ and $x_B^\mu = 1$, respectively.

Assume that the virtual edge $\varepsilon = st$ (oriented from s to t) in skel(μ) is an attachment of B, i.e., an attachment vertex $v \notin \{s,t\}$ of B lies in the expansion graph of ε. Let μ' be the neighbor of μ corresponding to ε and let ε' be the twin of ε in skel(μ') (also oriented from s to t); see Figure 8.12. Clearly, B is also important for μ' as ε' contains an attachment vertex of B while the attachment vertex v is not contained in ε'. In Step 1 we ensured that skel(μ') is embedded such that v (or the virtual edge containing v) shares a face with ε'. If this face lies to the left of ε' in skel(μ'), we say that v is an attachment *on the right side* of ε in skel(μ) (as in the example in Figure 8.12). Otherwise, if this faces lies to the right of ε', we say that v is an attachment *on the left side* of ε. For a bridge B with attachment vertex v we also say that the attachment ε is *right-sided* and *left-sided* if v lies on the right and left side of ε, respectively.

Assume B has an attachment on the right side of the virtual edge ε in the skeleton skel(μ) (as in Figure 8.12). Assume further that the skeleton skel(μ') of the corresponding neighbor is not flipped, i.e., $x_{\mu'} = 0$. Then B must be embedded into the face to the right of the cycle skel(μ), i.e., $x_B^\mu = 0$. Conversely, if the embedding of skel(μ') is flipped, i.e., $x_{\mu'} = 1$, then B must lie in the left face of skel(μ), i.e., $x_B^\mu = 1$. This necessary condition is equivalent to the equation $x_{\mu'} = x_B^\mu$. Similarly, if B has an attachment on the left side of ε, we obtain the inequality $x_{\mu'} \neq x_B^\mu$. We call the resulting set of equations and inequalities the *consistency constraints* of the bride B in μ. This leads to the second step of the algorithm.

Step 2: Consistency constraints. For every S-node μ compute the important ⓘ-bridges (for $i \in \{1,2\}$) and union bridges together with their attachments. For attach-

ments in virtual edges also compute whether they are left- or right-sided. Then add the consistency constraints of these bridges in μ to a global 2-Sat formula.

The consistency constraints are necessary but not sufficient as they do not ensure that no two alternating bridges of the same type are embedded into the same face. Consider an S-node μ with two important bridges B and B'. Assume these two bridges alternate (i.e., they have alternating attachments in the cycle skel(μ)). Embedding B and B' on the same side of skel(μ) yields a crossing between an edge in B and an edge in B'. Thus, if B and B' are both ①- or both ②-bridges, then they must be embedded to different side of skel(μ). In this case, we obtain the inequality $x_B^\mu \neq x_{B'}^\mu$. This inequality is called *planarity constraint*.

Step 3: Planarity constraints. For every S-node μ compute the pairs of important ①-bridges that alternate in skel(μ). Do the same for ②-bridges. For each such pair add the planarity constraint to the global 2-Sat formula.

Finally, we have to embed ⓘ-bridges belonging to the same union bridge into the same face. Let B be an ⓘ-bridge and let B' be the union bridge it belongs to. The *union-bridge constraint* of B in μ is the equations $x_B^\mu = x_{B'}^\mu$.

Step 4: Union-bridge constraints. For every S-node μ, add the union-bridge constraint of each important ⓘ-bridge to the global 2-Sat formula.

After Steps 2–4, the global 2-Sat formula is solved in linear time [APT79]. The solution determines for every P- and every R-node μ, whether the reference embedding of skel(μ) should be flipped or not, which completely fixes the embedding of the common graph G. Of course, there might be different solutions of the 2-Sat formula, yielding different embeddings. However, if one of these solutions yields a Sefe, then any of the solutions does [Ang+12]. Thus, one can simply take one solution and check whether it yields a Sefe (with union bridge constraints) or not.

Step 5: Final step. Test whether the given instance admits a Sefe with union bridge constraints assuming that the embedding of the common graph is fixed.

It remains to implement Steps 1–5 in linear time, which is done in the following. We first note that there are too many important bridges to be able to compute them in linear time (as required in Step 1 and Step 2). However, similar to Angelini et al. [Ang+12], we can show that many important bridges can be omitted without loosing the correctness of the algorithm, which leads to a linear-time implementation.

Too Many Bridges are Important

We start with the observation that computing all important union bridges for every P- and R-node of the SPQR-tree is actually a bad idea, as there may be $\Omega(n)$ bridges

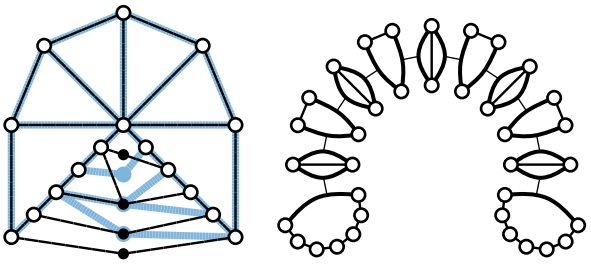


Figure 8.13: A graph with many bridges (left) each of which being important for every node of the SPQR-tree (right).

each being important in $\Omega(n)$ nodes. Thus, explicitly computing all of them would require $\Omega(n^2)$ time. Consider the graph G in Figure 8.13 whose SPQR-tree $\mathcal{T}$ (without Q-nodes) is a path. Let μ be one of the P-nodes (note that skel(μ) has two virtual edges and one normal edge) and let B be one of the bridges shown in Figure 8.13. Clearly, the expansion graphs of both virtual edges of skel(μ) contain at least one attachment vertex of B. Thus, B is important for μ and B has the two virtual edges of skel(μ) as attachments. As this may hold for a linear number of bridges, we get the above observation.

To resolve this issue, note that from the perspective of μ (in the above example), all bridges look the same in the sense that they have the same set of attachments. Intuitively, they thus lead to similar constraints and it seems to suffices to know only one of these bridges. In the following, we first show that omitting some of the bridges is indeed safe in the sense that the algorithm remains correct. Then we show how to compute the remaining bridges efficiently.

Omitting Some Important Bridges

In this section we show how to change the algorithm described above (Steps 1–5) slightly without changing its correctness. We say it is *safe* to do something if doing it does preserve the correctness of the algorithm. In particular, we show that it is safe to omit some of the important bridges. In the subsequent sections we then show that the remaining important bridges can be computed efficiently leading to an efficient implementation of all five steps.

Let G be the common graph and let $\mathcal{T}$ be its SPQR-tree. Let μ be an inner node of $\mathcal{T}$ and let B be a bridge that is important for μ with attachments $a_1, \ldots, a_\ell$ in skel(μ) (recall that each of the a_i is either a vertex or a virtual edge of skel(μ)). We call an attachment a_i (with $1 \leq i \leq \ell$) *superfluous*, if a_i is a vertex in skel(μ) such that B has another attachment a_j that is a virtual edge incident to the vertex a_i; see Figure 8.14a. The following lemma shows that the term "superfluous" is justified.

Lemma 8.12. *Omitting superfluous attachments is safe.*

Proof. There are two situation in which missing superfluous attachments might play a role. First, when we check a P- or R-node skeleton for a compatible embedding (Step 1).

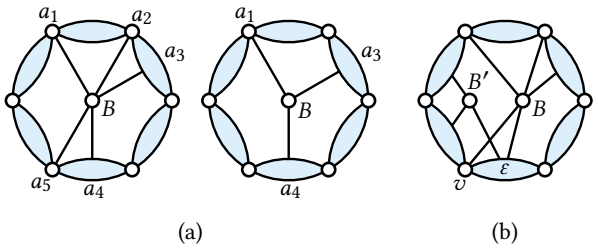


Figure 8.14: (a) The bridge B has five attachments $a_1, \ldots, a_5$; a_2 and a_5 are superfluous due to a_3 and a_4. (b) The bridges B and B' alternate only due to the superfluous attachment v of B. However, the consistency constraints synchronize B and B' as they have ε as common attachment.

Second, when we add the planarity constraints for alternating ⓘ-bridges (Step 3). Let v be the superfluous attachment of B and let ε be a virtual edge incident to v that is also an attachment of B in skel(μ). Concerning compatible embeddings, we have to make sure that skel(μ) admits an embedding where all attachments of B are incident to a common face. Clearly, v is incident to both faces the virtual edge ε is incident to. Thus, omitting the attachment v does not change anything.

Concerning the planarity constraints, we have to consider the case that B alternates with another bridge B' only due to the attachment v in B. This can only happen if B' has ε as attachment; see Figure 8.14b. However, then the consistency constraints (Step 2) either forces B and B' into different faces (if their attachment in ε lies on different sides of ε) and everything is fine, or they force B and B' to lie in the same face. In the latter case, the instance is clearly not solvable, which will be found out in Step 5. □

In the following we always omit superfluous attachments even when we do not mention it explicitly. Note that this retroactively changes the definition of important bridges slightly, i.e., a bridge B is important for a node μ if B has at least two (non-superfluous) attachments in skel(μ).

To show that we can omit sufficiently many important bridges to get a linear running time, we have to root the SPQR-tree $\mathcal{T}$. More precisely, we choose an arbitrary Q-node as the root of $\mathcal{T}$.

We categorize the important bridges in different types of bridges depending on their attachments. To this end, we first define different types of attachments. Let μ be a node of the (rooted) SPQR-tree and let B be an important bridge of μ with attachments $a_1, \ldots, a_\ell$. Recall that an attachment is either a vertex of a virtual edge of skel(μ). If a_i is a pole of skel(μ), we call it a *pole attachment*. If a_i is the parent edge of skel(μ), we call it *parent attachment*. All other attachments are called *child attachments*.

We say that the important bridge B is a *regular bridge* of μ if B has at least two child attachments. If B has only a single child attachment, it has either a parent attachment or one or two pole attachments (note that pole attachments are superfluous in the presence of a parent attachment). We call B *parent bridge* and *pole bridge* in the former

and latter case, respectively. Note that B must have at least one child attachment as it otherwise cannot be important.

As shown before, we cannot hope to compute all important bridges efficiently as there may be too many of them. Thus, we show in the following that omitting some important bridges is safe.

Lemma 8.13. *Let B be a parent bridge whose child attachment a is a virtual edge. Omitting another parent bridge or a pole bridge with child attachment a (while keeping B) is safe.*

Proof. Let B' be an other important bridge of μ and let a be the child attachment of B and B'. Let ε be the parent edge of skel(μ). In Step 1, the bridge B forces the embedding of skel(μ) to have a and ε on a common face. If B' is a parent bridge, it requires the same (and thus no additional) condition for the embedding of skel(μ). If B' is a pole bridge, it requires one of the poles (or both of them) to be incident to a common face with a. However, this is clearly the case if the parent edge ε (which is incident to both poles) shares a face with a. Hence, omitting B' does not change anything in Step 1.

We cannot argue separately for Steps 2–4 as they all contribute to the same global 2-Sat formula. We call the 2-Sat formula we get when omitting B' new. The formula we get when not omitting B' is called original. The straightforward way to prove that omitting B' is safe would be to show that the new and the original 2-Sat formulae are equivalent in the sense that they have the same solutions. However, this is not true as omitting B' can make the 2-Sat formula solvable whereas it was unsolvable before.

Assume the original 2-Sat formula is not solvable. Then the given instance does not admit a Sefe with union bridge constraints. In this case, Step 5 will never succeed no matter what we do in the steps before. Thus, we only have to argue for the case that the original 2-Sat formula is solvable. In this case, the only thing that can go wrong is that the new 2-Sat formula admits a solution that is not a solution in the original formula. This solution could then result in an embedding of the common graph G that does not admit a Sefe, whereas all solutions of the original 2-Sat formula lead to a Sefe.

To get a handle on this, we consider the conflict graph of a 2-Sat formula. First note that the formula we obtain is special in the sense that each constraint is either an equation or an inequality. We define the conflict graph to have a vertex for each variable and an edge connecting two vertices if there is an (in-)equality constraint between the corresponding variables. Recall that the 2-Sat formula has a variable x_μ for every P- or R-node μ indicating whether the embedding of skel(μ) has to be flipped or not. In the following we show that two such variables x_μ and x_η that are in the same connected component of the conflict graph of the original 2-Sat formula are also in the same connected component with respect to the new formula. This shows that every embedding resulting from a solution of the new formula also yields a solution of the original formula (assuming that the original formula has a solution at all).

First note that omitting an important bridge B' in an S-node μ has the effect that the vertex corresponding to the variable $x^\mu_{B'}$ is deleted in the conflict graph (for convenience we also denote the vertex by $x^\mu_{B'}$). Recall that the variable $x^\mu_{B'}$ represents the decision of whether B' is embedded into the left or the right face of skel(μ). We show that $x^\mu_{B'}$ is dominated by x^μ_B in the sense that every neighbor of $x^\mu_{B'}$ is also connected to x^μ_B by a path not containing $x^\mu_{B'}$. Thus, removing $x^\mu_{B'}$ does not change the connected components of the conflict graph.

Consider the consistency constraints of B' (Step 2). Let a be an attachment of B'. If a is a vertex in skel(μ), we do not get a consistency constraint there. If a is a virtual edge, it corresponds to a neighbor η of μ and we get the constraint $x_\eta = x^\mu_{B'}$ or $x_\eta \neq x^\mu_{B'}$. But then B has also a as attachment and thus we also have one of the two constraints $x_\eta = x^\mu_B$ or $x_\eta \neq x^\mu_B$.

For the planarity constraints (Step 3), first assume that B' is a parent bridge. Then B alternates with another bridge B'' if and only if B' alternates with B''. Thus, if we have the constraint $x_{B'} \neq x_{B''}$ we also have the constraint $x_B \neq x_{B''}$. If B' is a pole bridge, there might be a bridge B'' alternating with B' (yielding the constraint $x_{B'} \neq x_{B''}$), whereas B and B'' do not alternate. However, this can only happen if B'' has the parent edge ε of skel(μ) as attachment. Let η be the neighbor of μ corresponding to ε (i.e., the parent of μ). From Step 2 we then have the consistency constraint between x_η and $x^\mu_{B''}$ and also one between x_η and x^μ_B (B is a parent bridge). Thus, the conflict graph includes the path $x^\mu_B x_\eta x^\mu_{B''}$, which still exists after removing B'.

Finally, if B'' is the union-bridge including B', we loose the union-bridge constraint (Step 4) $x^\mu_{B'} = x^\mu_{B''}$ by omitting B'. Note that the attachments of B' are a subset of the attachments of B''. Thus, B'' has the child attachment a (which is a virtual edge). Let η be the neighbor of μ corresponding to a. Then we have a consistency constraint connecting $x^\mu_{B''}$ with x_η. Moreover, x_η is also connected to X^μ_B as a is an attachment of B. This yields a connection from x^μ_B to $x^\mu_{B''}$. □

This lemma gives rise to the following definition. Let B' be a pole bridge of μ with child attachment a that is a virtual edge in skel(μ). Let further B be a parent bridge of μ with child attachment a. We say that B' is *dominated* by B. If B' is a parent bridge of μ with child attachment a instead, we say that B and B' are *equivalent*. Note that being equivalent is clearly an equivalence relation. Lemma 8.13 shows that we can omit dominated pole bridges and all but one parent bridge for each equivalence class.

Computing the Remaining Important Bridges

In this section we show that all bridges that are not omitted due to Lemma 8.13 can be computed in linear time. Actually, we compute slightly more information, which we need in some intermediate steps. We for example never omit a parent bridge B in μ if B is regular in the parent of μ. We call such a bridge *semi-regular* in μ.

Moreover, consider an important bridge B of μ and let a be an attachment of B in skel(μ) that is a virtual edge. A vertex v of G that is not incident to a in skel(μ) is a *representative* of the attachment a if v is an attachment vertex of B and included in the expansion graph of a. If the attachment a is a vertex of skel(μ), we say that a is its own representative. Note that every attachment of B has at least one representative vertex. When we compute the important bridges of a node μ, we also compute their attachments in μ together with at least one representative for each attachment. Before we actually compute bridges, we provide some general tools.

Lemma 8.14. *After linear preprocessing time, the pole and parent attachments of a bridge in a given node together with a representative of each attachment can be computed in constant time.*

Proof. We first do some preprocessing. We define the *SPQR-vertex-tree* $\mathcal{T}$ to be the tree obtained from the SPQR-tree by removing its Q-nodes and attaching every vertex v of G as a leaf to the highest node that contains v in its skeleton. Note that this is the unique node that contains v in its skeleton but not as pole. For a non-pole vertex v in a skeleton skel(μ), we say that the leaf v (which is a child of μ) corresponds to v. Note that this makes sure that every child attachment in skel(μ) corresponds to a child of μ in $\mathcal{T}$.

Assume the vertices of G (i.e., the leaves of $\mathcal{T}'$) to be numbered according to a DFS-ordering (which we get in $O(n)$ time). We start by sorting the attachment vertices of each bridge according to this DFS-ordering. For all bridges $B_1, \dots, B_k$ together, this can be done in time $O\left(n + \sum_{i=1}^{k} |B_i|\right)$ as follows. To simplify the notation, we identify every bridge B_i with its set of attachment vertices. First, sort all pairs (v, B_i) with $v \in B_i$ (i.e., basically the disjoint union of the B_i) using bucket sort with buckets $1, \dots, n$ (for each vertex v, one bucket containing all pairs (v, B_i)). It then suffices to iterate over all these sorted pairs to extract the sets $B_1, \dots, B_k$ sorted according to this DFS-ordering.

For a node of $\mathcal{T}$, the leaves that are descendents of this node appear consecutively in the DFS-ordering. By going bottom-up in $\mathcal{T}$, we can compute for every node μ the leaf with the smallest and the leaf with the largest number among the descendents of μ. Thus, given a vertex v of G, we can decide in constant time whether v is a descendent of μ.

Now we answer the queries. Let B be a bridge that is important in a node μ. If the vertex with the smallest and the vertex with the largest number in B (which we can find in constant time as B is sorted) are both descendents of μ, all attachment vertices of B are descendents of μ. In this case B has neither a pole nor a parent attachment in μ. Otherwise, by looking at the first two and the last two elements in B we can distinguish the following two cases. (i) There is an attachment vertex v that is not a descendent of μ and not a pole of skel(μ). Then the parent edge of skel(μ) is an attachment of B in skel(μ) and v is a representative for this attachment. If B also has pole attachments

in skel(μ), they are superfluous and we can ignore them by Lemma 8.12. (ii) There is no such vertex v. Then all attachment vertices in B are descendents of μ except for maybe the two poles. In this case, the poles of skel(μ) that are attachments of B are among the first or last two vertices in B and thus we find all pole attachments of B in skel(μ). This concludes the proof. □

Lemma 8.15. *The regular bridges and their attachments together with a representative for each attachment can be computed in linear time.*

Proof. Let $\mathcal{T}$ be the SPQR-vertex-tree of G and assume again that the vertices in every bridge are sorted according to a DFS-order of the leaves in $\mathcal{T}$. We first show how to compute all nodes in which a given bridge B_i is regular in $O(|B_i|)$ time. Note that this implicitly shows that B_i has only $O(|B_i|)$ regular vertices.

The *lowest common ancestor (LCA)* of two vertices u and v is the highest node on the path between u and v. We denote it by $\mathrm{LCA}(u,v)$. Clearly, the vertices u and v are descendents of different children of $\mathrm{LCA}(u,v)$. Thus, if B_i has the attachment vertices u and v, then B_i has two different child attachments in $\mathrm{LCA}(u,v)$ with representatives u and v. Thus, $\mathrm{LCA}(u,v)$ is active. Conversely, if B_i is regular in a node μ, it has two different child attachments. Let u and v be representatives of these two attachments. Clearly, u and v are descendents of different children of μ and thus $\mu = \mathrm{LCA}(u,v)$ holds. Hence, the nodes in which B_i is regular are exactly the LCAs of pairs of attachment vertices in B_i.

To see that it is not necessary to consider all pairs of vertices in B_i, let u, v, and w with $u < v < w$ (according to the DFS-ordering) be contained in B_i. Then $\mathrm{LCA}(u,w) = \mathrm{LCA}(u,v)$ or $\mathrm{LCA}(u,w) = \mathrm{LCA}(v,w)$ holds for the following reason. If $\mu = \mathrm{LCA}(u,v) = \mathrm{LCA}(v,w)$, then u, v, and w are descendants of three different children of μ. Thus, $\mathrm{LCA}(u,w) = \mu$ also holds. Otherwise, assume $\mu = \mathrm{LCA}(u,v)$ is a descendant of $\eta = \mathrm{LCA}(v,w)$. Thus, from the perspective of η, the vertices u and v and the node μ are descendants of the same child whereas w is the descendant of a different child. Thus $\mathrm{LCA}(u,w) = \mathrm{LCA}(v,w) = \eta$. Hence, to compute all nodes in which B_i is regular, it suffices to compute the LCA for pairs of attachment vertices in B_i that are consecutive (with respect to the ordering we computed before). As the LCA can be computed in constant time after $O(n)$ preprocessing time [HT84], this gives us all regular bridges of all nodes of the SPQR-tree in overall $O(n + \sum_{i=1}^{k} |B_i|)$ time.

Given a node μ and a regular bridge B of μ, we are still lacking the attachments of B in skel(μ). We can get the parent and pole attachments using Lemma 8.14. Consider a child attachment a of B in skel(μ). Note that the attachment vertices of B that are descendants of a are consecutive with respect to the DFS-order. When choosing the first or last of these vertices, we get an attachment vertex v in B with the following properties: v is a descendent of the child of μ that corresponds to a and either $\mu = \mathrm{LCA}(u,v)$ for the predecessor u of v in B (ordered according to the

DFS-order) or $\mu = \text{LCA}(v, w)$ for the successor w of v. Thus, we actually already know a representative for every child attachment of B in skel(μ). To find for an attachment vertex v (the representative) the corresponding attachment in skel(μ), we need to find the child of μ that has the leaf v as a descendant. This can be done for all active bridges simultaneously by processing $\mathcal{T}$ bottom-up while maintaining a union-find data structure. As the sequence of union operations is known in advance, each union and find operation takes amortized $O(1)$ time [GT85]. Thus, it takes $O\left(n + \sum_{i=1}^{k} |B_i|\right)$ time in total. □

Lemma 8.16. *The semi-regular bridges and their attachments together with a representative for each attachment can be computed in linear time.*

Proof. By Lemma 8.15, we know for every node μ all the bridges that are regular in μ. We can assume that the regular bridges of a μ are stored as a list that is sorted according to an arbitrarily chosen order of the bridges (we just have to process the bridges in this order in the proof of Lemma 8.15). Moreover, we know all the attachments of B in skel(μ) together with a representative for each attachment.

Let a be a child attachment of B in skel(μ) with representative v. Let η be the child of μ corresponding to a. If η is a leaf, it actually must be the vertex v and there is nothing to do. Otherwise, v is a descendent of η and thus B has a child attachment in η. Moreover, it also has a parent attachment in η as it otherwise would not be regular in the parent μ of η. Thus, B is either regular or a parent bridge (and hence semi-regular) in η.

By processing all regular bridges of μ like that, we can build for η (and all other children of μ) a list of bridges that contains all semi-regular bridges and additionally some regular bridges. Note that building up this list takes constant time for each attachment of regular bridges. As we computed those attachments in linear time, there cannot be more than that many attachments.

To get rid of the bridges that are actually regular and not semi-regular, note that we can assume the list of semi-regular and regular bridges to be sorted (according to the arbitrary order of bridges we chose before). Thus, we can simply process this list and the list of regular bridges of η (which we computed using Lemma 8.15) simultaneously and throw out all those that appear in both lists. This leaves us with a list of semi-regular bridges for every node. In addition to that, we know an attachment vertex for every semi-regular bridge that is a representative of the child-attachment of that bridge. Thus, we also get the actual attachments in skel(η) in linear time using one bottom-up traversal as in the proof of Lemma 8.15. Moreover, we get the parent attachments using Lemma 8.14. □

Lemma 8.17. *The parent and pole bridges and their attachments together with a representative for each attachment can be computed in linear time when omitting dominated pole bridges and all but one parent bridge of each equivalent classes.*

Proof. Let $\mathcal{T}$ be the SPQR-vertex-tree of the common graph G.

We now process $\mathcal{T}$ bottom-up computing a list of bridges for each node μ. This list will contain all bridges that potentially are parent or pole bridges in μ and we denote it by $\text{pot}(\mu)$. More precisely, if a bridge B has an attachment vertex that is a descendent of μ and other attachment vertices that are not descendents of μ, then B is contained in $\text{pot}(\mu)$. As we do not have the time to tidy up properly, $\text{pot}(\mu)$ can also contain bridges whose attachment vertices are all descendents of μ.

We start with the leaves of $\mathcal{T}$, which are vertices of G. Let v be such a leaf. Then $\text{pot}(v)$ is the list of all bridges having v as attachment. By initializing $\text{pot}(v)$ with the empty list and then processing all bridges once, we can compute $\text{pot}(v)$ for all vertices in $O\left(n + \sum_{i=1}^{k} |B_i|\right)$ time. Now consider an inner node μ. We basically obtain the list $\text{pot}(\mu)$ by concatenating the lists of all children. But before concatenation we (partially) process these lists separately.

While processing μ (and the lists computed for the children of η) we want to answer for a given bridge B the following queries in constant time. First, is B regular in μ? Second, have we seen B already while processing μ? This can be done using timestamps. Assume we have one global array with an entry for each bridge. Before processing μ we increment a global timestamp and write this timestamp into the fields of the array corresponding to bridges that are regular in μ. While processing μ, we can then check in constant time whether the current timestamp is set for a given bridge B and thus whether B is regular. Setting up this array takes time linear in the number of regular bridges of μ and thus we have an overall linear overhead. We can handle the second query in constant time, analogously.

Now let μ be the node we currently process, let η be a child of μ and assume that $\text{pot}(\eta)$ is already computed. We process the list $\text{pot}(\eta)$; let B be the current bridge. By Lemma 8.14 we can check in constant time whether B has pole or parent attachments in μ. If not, all attachment vertices of B are descendents of μ and we remove μ form the list $\text{pot}(\eta)$ as B cannot be a parent or pole in μ or in any ancestor of μ.

Otherwise, B has a parent or a pole attachment in μ. We first check (in constant time) whether B is regular in μ. Assume it is regular and we see B the first time since processing μ (which we can also check in constant time as mentioned above). Then we simply skip B and continue processing $\text{pot}(\eta)$. If B is regular in μ and B occurred before while processing μ, it would be contained twice in the list $\text{pot}(\mu)$ when concatenating the lists of all children of μ. Thus we can remove this occurrence of B from $\text{pot}(\eta)$. Afterwards, we continue processing the remaining bridges in $\text{pot}(\eta)$.

Now assume that B is not regular. Then B is either a parent or a pole bridge. If B is a parent bridge, we store it as a parent bridge of μ. Note that we also know the attachments of B in $\text{skel}(\mu)$ (the parent edge and the attachment corresponding to the child η) and a representative for each of these attachments. Afterwards, we stop processing $\text{pot}(\eta)$ and continue with another child of μ. By stopping after processing B,

we might miss a bridge B' in $\text{pot}(\eta)$ that is also a pole or parent bridge of μ. However, the child attachment of B' would be the attachment corresponding to η and thus B' is in the same equivalence class as B (if B' is a parent bridge) or B' is dominated by B (if B' is a pole bridge). In both cases we can omit B'.

Finally, consider the case that B is a pole bridge in μ. We save B together with its attachments in $\text{skel}(\mu)$ (and their representatives) as pole bridge of μ. Then we remove B from $\text{pot}(\eta)$ and continue processing $\text{pot}(\eta)$. Removing B from $\text{pot}(\eta)$ has the effect that B does not occur when processing an ancestors of μ. Thus, we have to show that B is not a pole or parent bridge in one of these ancestors. Consider an ancestor τ of μ. Then either all attachment vertices of B are descendents of τ and B is neither pole nor parent bridge for τ. Otherwise, the only attachment vertex of B that is not a descendent of τ is s (without loss of generality). However, the virtual edge in $\text{skel}(\tau)$ containing all attachment vertices of B is then incident to s and thus the attachment s is superfluous. Hence, we can omit the bridge B in $\text{skel}(\tau)$ by Lemma 8.12.

It remains to show that the above procedure runs in linear time. First note that we add elements to lists $\text{pot}(\cdot)$ only in the leaves of $\mathcal{T}$. As we add each bridge B to exactly $|B|$ such bridges, the total size of these lists is linear. Assume we are processing a list $\text{pot}(\eta)$ and let B be a bridge we delete after processing it. Then we can ignore the (constant) running time for processing B, as we have only linearly many such deletion operations. Otherwise, B is a regular bridge that we see the first time or it is a parent bridge. The former case happens only as many times as there are regular bridges of μ (which is overall linear). The latter case happens at most once for each child of μ as we stop processing $\text{pot}(\eta)$ afterwards. Hence, the overall running time is linear. □

Linear Time Implementation of Steps 1–5

Lemma 8.18. *Step 1 can be performed in linear time.*

Proof. Recall that Step 1 consist of computing the important union bridges for P- and R-nodes. For every P- and R-node μ we then have to test whether $\text{skel}(\mu)$ admits a compatible embedding, i.e., whether $\text{skel}(\mu)$ can be embedded such that the attachments of each important bridge of μ share a face.

For each node μ, we compute the regular and semi-regular bridges using Lemma 8.15 and Lemma 8.16, respectively. We moreover compute some of the pole and parent bridges using Lemma 8.14. In this way we of course miss some important bridges but we know by Lemma 8.13 that it is safe to do so. Thus, we can focus on computing compatible embeddings.

Let μ be a node of the SPQR-tree and assume that μ is a P-node. Each of the two vertices s and t in $\text{skel}(\mu)$ is incident to every face. If an important bridge B has s or t as attachment, this attachment does not constrain the embedding of $\text{skel}(\mu)$ (it shares a face with all other attachments of B if and only if all other attachments share a face).

Thus, we can assume that only the virtual edges of skel(μ) are attachments. If B has three (or more) attachments in skel(μ), it is impossible to find a compatible embedding as every face of skel(μ) is incident to only two virtual edges. It remains to deal with the case where every bridge has two virtual edges as attachment. We build the conflict graph with one vertex $v(\varepsilon)$ for every virtual edge ε and an edge between two such vertices $v(\varepsilon_1)$ and $v(\varepsilon_2)$ if and only if there is a bridge with attachments ε_1 and ε_2. It is not hard to see that skel(μ) admits a compatible embedding if and only if this conflict graph has maximum degree 2 and either contains no cycle or is a Hamiltonian cycle.

If μ is an R-node, its skeleton is triconnected and therefore has a fixed planar embedding. To test whether the embedding of skel(μ) is compatible, we need to check for every bridge B, whether there is a face incident to all its attachments. We consider the graph skel$'(\mu)$ obtained from skel(μ) by subdividing every edge and inserting a vertex into every face that is connected to all incident vertices. We denote the new vertex created by subdividing the virtual edge ε by $v(\varepsilon)$ and the new vertex inserted into the face f by $v(f)$. For a vertex v that already existed in skel(μ) we also write $v(v)$. For a bridge B with attachments $a_1, \dots, a_k$, we need to test whether there is a face f such that for every pair $v(a_i)$ and $v(a_j)$ the path $v(a_i)v(f)v(a_j)$ is contained in skel$'(\mu)$. To make sure that all paths of length 2 between $v(a_i)$ and $v(a_j)$ include a vertex $v(f)$ corresponding to a face f, we subdivide every edge twice except for those edges incident to a vertex $v(f)$ corresponding to a face.

As skel$'(\mu)$ is planar, we can use the data structure by Kowalik and Kurowski [KK03] that can be computed in linear time and supports shortest path queries for constant distance in constant time. More precisely, for any constant d, there exists a data structure that can test in $O(1)$ whether a pair of vertices is connected by a path of length d. If so, a shortest path is returned. We first rule out some easy cases.

If there is an attachment a_i that is a virtual edge in skel(μ), the vertex $v(a_i)$ has only four neighbors in skel$'(\mu)$. Thus, we get the two faces incident to a_i in constant time and can check in $O(k)$ time whether one of them is incident to every attachment of B. We thus assume that all attachments are vertices. If one of these vertices is adjacent to three or more others, there cannot be a compatible embedding. Thus, we either find (in $O(k)$ time) a pair of non-adjacent attachments a_i and a_j or there are only two or three pairwise adjacent attachments. As the latter case is easy, we can assume that we have non-adjacent attachments a_i and a_j. As skel(μ) is triconnected, a_i and a_j are incident to at most one common face f. Thus, there is only one path $v(a_i)v(f)v(a_j)$ of length 2 from a_i to a_j in skel$'(\mu)$, which gives us f in constant time. Then it remains to check whether all other attachments are incident to f, which takes $O(k)$ time. Doing this for every bridge yields a linear-time algorithm testing whether an R-node skeleton admits a compatible embedding. □

Lemma 8.19. *Step 2 can be performed in linear time.*

Proof. Recall that Step 2 consist of three parts. First we have to compute the important

ⓘ-bridges of S-nodes together with their attachments. For each attachment we then have to test whether it is left- or right-sided. Finally, we have to add the consistency constraints.

As for Step 1, we use Lemma 8.15, Lemma 8.16, and Lemma 8.17 to compute all important ⓘ-bridges together with their attachments. We actually compute these important bridges not only for the S-nodes but also for P- and R-nodes. We use this additional information to compute which attachments are left- and which are right-sided.

Note that the final step of adding the consistency constraints to a global 2-Sat formula is trivial. Thus, it remains to show that we can compute for every attachment whether it is left- or right-sided.

Let μ be a node of the SPQR-tree $\mathcal{T}$. We iterate over all important bridges of μ (except those we omitted). For every bridge B we iterate over all attachments in μ and for every virtual edge ε among those attachments, we append B to the list bridges(ε). Afterwards, bridges(ε) contains all important (but not omitted) bridges of μ that have ε as attachment. Note that we can assume that the bridges in bridges(ε) are sorted according to an arbitrary but fixed order of the bridges.

Let μ be an S-node with virtual edge ε in skel(μ). Let further μ' be the neighboring P- or R-node corresponding to ε and let ε' be the virtual edge in skel(μ') corresponding to the S-node μ. For every bridge B in bridges(ε) we want to know whether the attachment ε is left- or right-sided. To this end we iterate over the lists bridges(ε) and bridges(ε') simultaneously. For every bridge B in bridges(ε) there are two different cases. Either B also occurs in bridges(ε') or it does not. It is not hard to see that the latter can only happen if B is in μ' a pole or parent bridge that was omitted.

If B also occurs in bridges(ε'), we know from Step 1 that skel(μ') has a (unique) face incident to all attachments of B in skel(μ'). In particular, this face is either the right or the left face of ε' and thus we immediately know whether the attachment ε of B in skel(μ) is left- or right-sided.

It remains to consider the case where B occurs in bridges(ε) but not in bridges(ε'). If μ' is the parent of μ, the bridge B must be a pole- or parent bridge in μ' with child attachment ε'. As in the proof of Lemma 8.18, we can find the (unique) face incident to ε' and one of the poles. As B has to lie in this face we know whether it lies to the right or to the left face of ε' in skel(μ') and thus we know whether the attachment ε in skel(μ) is left- or right-sided.

If μ' is a child of μ, then B cannot be regular in μ as otherwise B would be semi-regular in μ' and thus contained in bridges(ε') (which is not the case we consider). Hence B is either a pole or a parent bridge (recall that semi-regular bridges are also parent bridges). Thus, B is a pole or parent bridge in μ with child attachment ε. By Lemma 8.13, we can omit all but a constant number of such bridges with ε as child attachment. Thus, we can assume that bridges(ε) contains only a constant number

of bridges that do not occur in bridges(ε'). For these bridges we allow a running time linear in the size of skel(μ'). As μ is the unique parent of μ' this happens only a constant number of times for μ' and thus takes overall linear time.

Let B be such a bridge and let v be the attachment vertex of the common graph G representing the child attachment ε of B in skel(μ). We show how to find a child attachment of B in skel(μ') in $O(|\operatorname{skel}(\mu')|)$ time. Then we can (as in the cases before) find in constant time which face incident to ε' contains B in skel(μ') and we are done. As before we assume to have a DFS-ordering on the leaves of the SPQR-vertex-tree. Then the leaves that a descendents of an inner node form an interval with respect to this order. These intervals can be easily computed in linear time by processing the SPQR-vertex-tree bottom up once. Afterwards, we can check in constant time whether a vertex v is the descendent of an inner node. Hence we can check in $O(|\operatorname{skel}(\mu')|)$ time which child of μ' is an ancestor of v and thus which virtual edge or vertex in skel(μ') represents v. This yields the child attachment of B in skel(μ') and we are done. □

Lemma 8.20. *Step 3 can be performed in linear time.*

Proof. Let μ be an S-node. Note that every important bridge in μ may alternate with a linear number of important bridges. Thus, there are instances where the planarity constraints have quadratic size. In the following we describe how to compute constraints that are equivalent to the planarity constraints but have linear size. We only consider ①-bridges; for ②-bridges, the same procedure can be applied.

We define the graph H as follows. We start with $H = \operatorname{skel}(\mu)$ and subdivide every edge once. Thus, H has a vertex $v(a)$ for each attachment a in skel(μ). For every bridge B with attachments $a_1, \dots, a_k$, we add a *bridge vertex* $v(B)$ and connect it to the vertices $v(a_1), \dots, v(a_k)$. When using the term *cycle of* H, we refer to the subgraph of H one obtains by removing the bridge vertices.

Assume we have a planar embedding of H. Then, every bridge vertex lies on one of the two sides of the cycle and no two bridges on the same side of the cycle alternate. Conversely, an assignment of the bridges to the two sides of the cycles such that no two bridges alternate yields a planar embedding. The choices the planarity constraints leave are thus equivalent to the embedding choices of H.

As H is biconnected, the embedding choices consist of reordering parallel edges in P-nodes and mirroring R-nodes of the SPQR-tree $\mathcal{T}_H$ of H. Let η be a P-node in $\mathcal{T}_H$. If the embedding of skel(η) determines on which side of the cycle a bridge B lies, then B is clearly not alternating with any other bridge. For an R-node η of $\mathcal{T}_H$, fixing the embedding of skel(η) to one of the two flips determines the side for some of the bridges. We create a new binary variable x_η with the interpretation that $x_\eta = 0$ if skel(η) is embedded according to a reference embedding and $x_\eta = 1$ if the embedding is flipped. For a bridge B whose side is determined by the embedding of skel(η) we

can then add the constraint $x_\eta = x_B^\mu$ or $x_\eta \neq x_B^\mu$ (depending on whether the reference embedding of skel(η) fixes B to the left or right side of the cycle).

It is not hard to see that one can compute for each R-node η the brides whose side is determined by the embedding of skel(η) in overall linear time in the size of H. Thus, we get the above constraints (which are equivalent to the planarity constraints) in $O(|H|)$ time. Clearly, the size of H is linear in the size of skel(μ) plus the total number of attachments of important bridges in μ. Thus, we get an overall linear running time. □

Lemma 8.21. *Step 4 can be performed in linear time.*

Proof. To add the union-bridge constraints, we have to group the ⓘ-bridges that are important in an S-node μ according to their union bridges. To this end, we once create a global array A with one entry $A[B']$ for each union bridge B' (which we can access in constant time). Consider an ⓘ-bridge B that is important in μ (and was not omitted). Let B' be the union bridge containing B (we can get B' in constant time as every ⓘ-bridge is contained in only one union bridge). If the entry $A[B']$ was not modified while processing μ so far, we clear $A[B']$ (which might contain something from previous nodes) and set $A[B']$ to be a list containing only B. If $A[B']$ was already modified, we append B to the list $A[B']$. We can keep track of which entries of A were already modified by using timestamps.

For every union bridge B' that contains an important ⓘ-bridge, the entry $A[B']$ holds a list of all important ⓘ-bridges of μ that belong to B'. For each of these lists we add the union-bridge constraint for every pair of consecutive ⓘ-bridges. The transitivity enforces all pairwise union-bridge constraints (although not explicitly stated). Clearly, this procedure takes linear time in the number of important ⓘ-bridges. □

For Step 5, assume the embedding of the biconnected common graph G is fixed. We have to test whether this embedding of G can be extended to a Sefe that satisfies the union-bridge constraints. Thus, we basically have to assign each union bridge to a face of G such that no two ①-bridges and no two ②-bridges alternate.

We first distinguish three different types of union bridges. Let B be a union bridge. A face f of G is feasible for B if all attachment vertices of B are incident to f. We say that B is *flexible* if it has at least three feasible faces. We say that B is *binary* if B has two feasible faces and *fixed* if it has only one feasible face. If there is a bridge that has no feasible face then the instance is obviously not solvable.

The overall strategy for Step 5 is the following. We first determine which union bridges are flexible, binary, and fixed, respectively. We first assign the fixed union bridges to their faces. For the binary union bridges, we can encode the decision for one of the two possible faces with a binary variable. For two union bridges with a common feasible face that include alternating ⓘ-bridges (for the same $i \in \{1,2\}$), we have to make sure that they are not embedded into the same face. Note that these kind

of conditions are very similar to the planarity constraints we had in Step 3, which again leads to a 2-Sat formula. Any solution of this formula induces an assignment of the binary union bridges to faces. Finally, we check whether the flexible union bridges can be added. For this to work we have to show that this final step of assigning the flexible bridges is independent from the solution we chose for the 2-Sat formula. We obtain the following lemma.

Lemma 8.22. *Step 5 can be performed in linear time.*

Proof. Let B be a flexible union bridge and assume all binary and fixed union bridges are already assigned to faces. We first show the following. If B cannot be embedded into one of its feasible faces (due to alternating ⓘ-bridges), then B cannot be embedded into this face even when omitting all binary union bridges. This shows that the above strategy of first assigning the fixed, then the binary, and finally the flexible union bridges to faces is correct. Afterwards we show how to do it in linear time.

As the union bridge B is flexible, it has at least three feasible faces. As the common graph G is biconnected, B can have only two attachment vertices; let u and v be these attachment vertices. Moreover, u and v must be the poles of a P-node μ of the SPQR-tree of G. Let $\varepsilon_1, \dots, \varepsilon_k$ be the virtual edges of $\text{skel}(\mu)$ appearing in this order and let f_i be the face between ε_i and ε_{i+1} (subscripts are considered modulo k). The feasible faces of B are exactly the faces $f_1, \dots, f_k$.

Assume B cannot be assigned to the feasible face f_i, i.e., an ⓘ-bridge contained in B alternates with a ⓘ-bridge belonging to another union bridge B' that was assigned to f_i. As u and v are the only attachment vertices of B, B' must have attachment vertices u' and v' with $u', v' \notin \{u, v\}$ that belong to the expansion graphs of ε_i and ε_{i+1}, respectively. Then u' and v' can share only a single face, namely f_i, and thus B' is a fixed union bridge. This shows the above claim and thus it remains to show how to implement the procedure in linear time.

Let B be an arbitrary union bridge. We show how to detect whether B is flexible, binary, or fixed in $O(|B|)$ time. For B to be flexible, it must have only two different attachment vertices. This can be easily tested in $O(|B|)$ time. If B has only two attachment vertices u and v, we need to test whether u and v are the poles of a P-node in the SPQR-tree of G. We show how this can be done in constant time. To this end, let $\mathcal{T}$ be the SPQR-vertex-tree of G. Assume μ is a P-node with poles u and v. Let η be the parent of μ. Then $\text{skel}(\eta)$ contains both vertices u and v but at most one of them as pole. Assume without loss of generality that u is not a pole of $\text{skel}(\eta)$. Then the leaf u of $\mathcal{T}$ is a child of η and thus we find η in constant time (together with the vertex u in $\text{skel}(\eta)$). If v is not a pole of $\text{skel}(\eta)$, we find v in $\text{skel}(\eta)$ in the same way, otherwise v is a pole and we also get v in $\text{skel}(\eta)$ in constant time (by checking both poles). As $\text{skel}(\eta)$ is a planar graph we can get the virtual edge between the two vertices u and v in constant time via a shortest-path data structure [KK03]. Thus, we also find the corresponding child μ having u and v as poles. Hence, we can either find the P-node μ

with poles u and v in constant time (implying that B is flexible) or we can conclude that such a P-node does not exists (implying that B is not flexible).

Next we determine whether B is binary or fixed. First note that the bridge B (that is not flexible) is binary if and only if there exists an S-node μ such that every attachment vertex of B is a vertex in skel(μ). This can be tested in $O(|B|)$ time using the SPQR-vertex-tree. Assume μ is the S-node such that every attachment vertex of B is a vertex in skel(μ). We can handle the case where B has only the two poles of skel(μ) as attachment vertex analogously to the case above (about flexible bridges) except that μ is an S-node instead of a P-node. Thus, assume that v is an attachment vertex of B that is not a pole of skel(μ). Then we can find μ in constant time as it is the parent of the leaf v in $\mathcal{T}$. Every other attachment vertex in B is either also a child of μ or a pole of skel(μ), which we can check in constant time per attachment vertex. Thus, we can detect in $O(|B|)$ time whether B is binary or fixed.

At this point we know which bridges are binary and which are flexible. All remaining bridges are either fixed or do not have a feasible face at all, which implies that there is no Sefe. We show for such a bridge B how we can assign it to its unique feasible face or decide that such a face does not exist. Recall from Lemma 8.18 that we can compute in constant time a face that is shared by a given pair of vertices (or conclude that such a face does not exist). If B has only two attachment vertices u and v, then we can either find the unique feasible face of B or decide that B has no feasible face in constant time.

We can thus assume that B has at least three attachment vertices. Let u, v, and w be three attachment vertices of B. In constant time, we find a face $f_{u,v}$ that is incident to u and v. Analogously we find faces $f_{u,w}$ and $f_{v,w}$. There are two different cases. If there is a pair of vertices among u, v, and w that shares only a single face, then one of the faces $f_{u,v}$, $f_{u,w}$, or $f_{u,w}$ is the only possible feasible face of B. We can check that in $O(|B|)$ time. Otherwise, assume there is a face $f \notin \{f_{u,v}, f_{u,w}, f_{u,w}\}$ that is feasible for B. Then u and v are commonly incident to at least two different faces (namely f and $f_{u,v}$) and thus $\{u, v\}$ is a separating pair of an edge. The same holds for u and w and for v and w. In this case there must exist a node μ in the SPQR-tree of G such that skel(μ) contains the triangle u, v, w. Note that we can find this node as we did before for the flexible and binary bridges.

If μ is an S-node, B must contain another attachment vertex x (otherwise B is binary). Then x is contained in the expansion graph of one of the three virtual edges in skel(μ). Assume without loss of generality that x belongs to the expansion graph of uv. Then w and x share only a single face (otherwise w and x would be a separating pair which contradicts the fact that skel(μ) is a triangle). Thus, we can find the desired face f by finding a common face of x and w in constant time. Of course one then needs to check if this face is actually incident to each attachment vertex in B (B has no feasibly face if not).

It remains to consider the case that μ is an R-node. First test whether the triangle u, v, w forms a face in skel(μ). If so, this face is unique and thus we know the only potentially feasible face of B. Note that this gives us only the face in skel(μ). However, one can easily compute a mapping from the faces of skeletons to the faces in the actual graph in linear time in the size of G (this has to be done only once for all bridges).

To conclude, we now ensured that every union bridge that is neither flexible nor binary is fixed and we assigned the fixed union bridges to their unique feasible faces. Let us continue with the binary bridges. For every binary bridge B we already computed the S-node μ containing all the attachment vertices of B. Note that this already gives us the two possible faces in which B can be embedded (of course we again have to translate from faces in a skeleton to faces in G).

When assigning the binary bridges to faces, we have to make sure that no two ⓘ-bridges alternate. This can be ensured using a 2-Sat formula as for Step 3. As before we can compute and solve this 2-Sat formula in linear time. Thus, it remains to add the flexible bridges.

Let μ be a P-node of the SPQR-tree of G and let s and t be the poles of skel(μ). Let further $\varepsilon_1, \ldots, \varepsilon_k$ be the virtual edges of skel(μ) and let the faces $f_1, \ldots, f_k$ be defined as before. Assume we still know the important bridges for μ from the previous steps. Assume the union flexible bridge B contains only ①-bridges. Clearly, we can embed B into f_i if and only if there is no ①-bridge with attachments ε_i and ε_{i+1}. The analogous statement holds if B contains only ②-bridges. If B contains both, ①- and ②-bridges, we can embed it into f_i if and only if there is neither a ① nor a ② bridge with attachments ε_i and ε_{i+1}. Thus, we can check in $O(|\text{skel}(\mu)|)$ time for an appropriate face for B. Note that we cannot afford this amount of time for every flexible bridge with attachments s and t. However, consider two such bridges B and B' as equivalent in the sense that B contains ①- and ②-bridges if and only if B' contains ①- and ②-bridges, respectively. Then B and B' can be assigned to the same face. Thus, we have to spend $O(|\text{skel}(\mu)|)$ time only a constant number of times for each P-node. This concludes the proof. □

Theorem 8.7. *Sefe with union bridge constraints can be solved in linear time if the common graph is biconnected.*

8.4 Edge Orderings and Relative Positions

In this section, we consider a Sefe instance $(G^①, G^②)$ that has common P-node degree 3 and simultaneous cutvertices of common degree at most 3. Recall that $(G^①, G^②)$ admits a simultaneous embedding if and only if $G^①$ and $G^②$ admit planar embeddings that have consistent edge orderings and consistent relative positions on the common graph. We show how to address both requirements (more or less) separately, by formulating necessary and sufficient constraints using equations and inequalities on

Boolean variables. Moreover, we show how to incorporate equations and inequalities equivalent to block-local common-face constraints. Together with the preprocessing algorithms from the previous sections, this leads to a polynomial time algorithm for instances with P-node degree 3 and simultaneous cutvertices of common degree at most 3.

Before we can follow this strategy, we need to address one problem. The relative position of a component H' of G with respect to another connected component H, denoted by $\text{pos}_H(H')$, is the face of H containing H'. However, the set of faces of H depends on the embedding of H. To be able to handle relative positions independently from edge orderings, we need to express the relative positions independently from faces.

This is done in the following section. Afterwards, we show how to enforce consistent edge orderings (Section 8.4.2), block-local common-face constraints (Section 8.4.3), and consistent relative positions (Section 8.4.4). Finally, we conclude in Section 8.4.5.

Before we start, we need one more definition. Assume we have a set X of binary variables such that every variable $x \in X$ corresponds to a binary embedding choice in a given graph G. Let $\alpha\colon X \to \{0,1\}$ be a *variable assignment*. We say that an embedding of G *realizes* the assignment α, if the embedding decision in G corresponding to a variable $x \in X$ fits to the value $\alpha(x)$. Note that not every variable assignment can be realized as the embedding choices can depend on each other.

8.4.1 Relative Positions with Respect to a Cycle Basis

A *generalized cycle* C in a graph H is a subset of its edges such that every vertex of H has an even number of incident edges in C. The *sum* $C \oplus C'$ of two generalized cycles is the symmetric difference between the edge sets, i.e., an edge e is contained in $C \oplus C'$ if and only if it is contained in C or in C' but not in both. The resulting edge set $C \oplus C'$ is again a generalized cycle. The set of all generalized cycles in H is a vector space over $\mathbb{F}_2$. A basis of this vector space is called *cycle basis* of H.

Instead of considering the relative position $\text{pos}_H(H')$ of a connected component H' with respect to another component H, we choose a cycle basis $\mathcal{C}$ of H and show that the relative positions of H' with respect to the cycles in $\mathcal{C}$ suffice to uniquely define $\text{pos}_H(H')$, independent from the embedding of H. We assume H to be biconnected. All results can be extended to connected graphs by using a cycle basis for each block.

Let $C_0, \dots, C_k$ be the set of facial cycles with respect to an arbitrary planar embedding of H. The set $\mathcal{C} = \{C_1, \dots, C_k\}$ obtained by removing one of the facial cycles is a cycle basis of G. A cycle basis that can be obtained in this way is called *planar cycle basis*. In the following we assume all cycle bases to be planar cycle bases. Moreover, we consider all cycles to have an arbitrary but fixed orientation. The binary variable $\text{pos}_C(p)$ represents the relative position of a point p with respect to a cycle C, where

$\text{pos}_C(p) = 0$ and $\text{pos}_C(p) = 1$ have the interpretation that p lies to the right and left of C, respectively.

Theorem 8.8. *Let H be a planar graph embedded on the sphere, let p be a point on the sphere, and let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be an arbitrary planar cycle basis of H. Then the face containing p is determined by the relative positions $\text{pos}_{C_i}(p)$ for $1 \leq i \leq k$.*

Proof. Let f be a face and let C be the corresponding facial cycle. We assume without loss of generality that $C = C_1 \oplus \cdots \oplus C_\ell$ holds. We show that the point p belongs to the face f if and only if $\text{pos}_{C_i}(p) = \text{pos}_{C_i}(f)$ holds for $1 \leq i \leq \ell$. Obviously, if $\text{pos}_{C_i}(p) \neq \text{pos}_{C_i}(f)$ holds for one of the basis cycles C_i, then C_i separates p from f and thus p cannot belong to f.

Conversely, we have to show that there is no point lying on the same sides of the cycles C_i for $1 \leq i \leq \ell$ not belonging to f. To this end we define the *position vector* $\text{pos}(p) = (\text{pos}_{C_1}(p), \ldots, \text{pos}_{C_\ell}(p))$ of a point p. We show that there is not point outside f having the same position vector as the points inside f. Consider how the position vector of a point p changes when moving it around. First, all points inside f have the same position vector. Second, when p does not lie in f and crosses an edge e while moving it, then e is either contained in zero or in two of the basis cycles $C_1, \ldots, C_\ell$. This comes from the facts that $C = C_1 \oplus \cdots \oplus C_\ell$ holds, that e is not contained in C and that our cycle basis is planar. In the former case the position vector does not change, in the latter case exactly two values toggle. No matter which case applies, the parity of the number of entries with the value LEFT does not change, that is this number either remains odd or even. Thus, this parity is the same for all points outside of f. Finally, when p moves from inside f to the outside of f (or the other way round), it has to cross an edge e contained in the cycle C. Since e has to be contained in exactly one of the cycles $C_1, \ldots, C_\ell$, exactly one entry in the position vector $\text{pos}(p)$ changes from LEFT to RIGHT or vice versa. Thus the parity of the number of entries with the value LEFT changes. It follows, that for every point not contained in f this parity differs from the parity of the points in f. Thus also the position vector must differ, which concludes the proof. □

To represent the relative position of one connected component H with respect to another connected component H', it thus suffices to consider the relative positions of H with respect to cycles in a cycle basis of H'. However, there is one case for which we have a slightly stronger requirement. To motivate this, consider the following example; see also Figure 8.15.

Consider the graph $G^{①}$ containing the common graph G with connected components H and H'. Let $\mathcal{C}$ be a cycle basis of H'. Let further v be a vertex of H' that is cutvertex of $G^{①}$ separating H from H' and let $C \in \mathcal{C}$. If v lies to the right of C in a given embedding of $G^{①}$, then H also lies to the right of C. Conversely, if v lies to the left of C, then H lies to the left of C. However, requiring for every cycle $C \in \mathcal{C}$ (that does

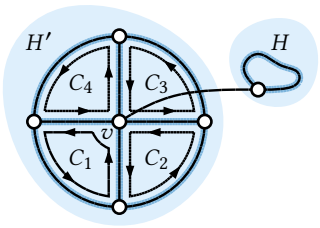


Figure 8.15: The exclusive edge connecting H to the vertex v of H' requires H to lie to the left of exactly one of the cycles $C_1, \ldots, C_4$. This cannot be expressed using only equations or inequalities. If, however, the cycle basis contained the facial cycle of the outer face of H', it would be sufficient to require that H and v lie on the same side of this cycle, which can be expressed using an equation.

not contain v) that v and H lie on the same side of C does not ensure that H lies in a face of H' that is incident to v. Figure 8.15 shows a somewhat degenerate example, were v is contained in every cycle of C.

Thus, the relative positions of v with respect to all cycles in C (that do not contain v) do not uniquely determine a face of $H' - v$. To resolve this issue, we add further cycles of H' to C. More precisely, an *extended cycle basis* of H' is a set of cycles C in H' such that C includes a cycle basis of H' and a cycle basis of $H' - v$ for every vertex v of H'.

Note that one can for example obtain an extended cycle basis of H' as follows. First choose an embedding of H' and start with the corresponding planar cycle basis for C. For every vertex v, consider the induced embedding of $H' - v$ and add to C all cycles in the corresponding planar cycle basis of $H' - v$ that are not already contained in C. It directly follows that an extended cycle basis has $O(n^2)$ size and can be computed in $O(n^2)$ time. Moreover, we get the following lemma.

Lemma 8.23. *Let H be an embedded planar graph and let C be an extended cycle basis of H. If a vertex v of H is not incident to f of H, then C contains a cycle C (not containing v) such that* $\mathrm{pos}_C(v) \neq \mathrm{pos}_C(f)$.

Proof. Consider the graph $H - v$ together with the embedding induced by the embedding of H. Let f_v be the face of $H - v$ that contains v in the embedding of H. As v is not incident to f, we have $f_v \neq f$. By Theorem 8.8, the cycle basis of $H - v$ (and thus C) must contain a cycle C such that $\mathrm{pos}_C(f_v) \neq \mathrm{pos}_C(f)$. □

If we refer to a cycle basis in one of the following sections, we always assume to actually have an extended cycle basis.

8.4.2 Consistent Edge Orderings

We first assume that the graphs $G^{①}$ and $G^{②}$ are biconnected and then show how to extend our approach to exclusive cutvertices and simultaneous cutvertices of common degree 3.

Biconnected Graphs

Let $G^{①}$ and $G^{②}$ be biconnected planar graphs. There exists an instance of Simultaneous PQ-Ordering that has a solution if and only if $G^{①}$ and $G^{②}$ admit embeddings with consistent edge ordering; see Chapter 5. This solution is based on the *PQ-embedding representation*, an instance of Simultaneous PQ-Ordering representing all embeddings of a biconnected planar graph. We describe this embedding representation and show how to simplify it for instances that have common P-node degree 3.

For each vertex $v^{①}$ of $G^{①}$, the PQ-embedding representation, denoted by $D(G^{①})$, contains the *embedding tree* $T(v^{①})$ having a leaf for each edge incident to $v^{①}$, representing all possible orders of edges around $v^{①}$. For every P-node $\mu^{①}$ in the SPQR-tree $\mathcal{T}(G^{①})$ that contains $v^{①}$ in $\text{skel}(\mu^{①})$ there is a P-node in $T(v^{①})$ representing the choice to order the virtual edges in $\text{skel}(\mu^{①})$. Similarly, for every R-node $\mu^{①}$ of $G^{①}$ containing $v^{①}$ in its skeleton, there is a Q-node in $T(v^{①})$ whose flip corresponds to the flip of $\text{skel}(\mu^{①})$. As the orders of edges around different vertices of $G^{①}$ cannot be chosen independently from each other, so called *consistency trees* are added as common children to enforce Q-nodes stemming from the same R-node in $\mathcal{T}(G^{①})$ to have the same flip and P-nodes stemming from the same P-node to have consistent (i.e., opposite) orders. Every solution of the resulting instance corresponds to a planar embedding of $G^{①}$ and vice versa; see Chapter 5.

As we are only interested in the order of common edges, we modify $D(G^{①})$ by projecting each PQ-tree to the leaves representing common edges. As $G^{①}$ and $G^{②}$ have common P-node degree 3, all P-nodes of the resulting PQ-trees have degree 3 and can be assumed to be Q-nodes representing a binary decision. We call the resulting instance *Q-embedding representation* and denote it by $D(G^{①})$.

Let $\mu^{①}$ be an R-node of the SPQR-tree $\mathcal{T}(G^{①})$ whose embedding influences the ordering of common edges around a vertex. Then the Q-embedding representation contains a consistency tree consisting of a single Q-node representing the flip of $\text{skel}(\mu^{①})$. We associate the binary variable $\text{ord}(\mu^{①})$ with this decision.

For a P-node $\mu^{①}$ we get a similar result. Let $u^{①}$ and $v^{①}$ be the poles of $\mu^{①}$. If the consistency tree enforcing a consistent decision in the embedding trees $T(u^{①})$ and $T(v^{①})$ has degree 3, its flip represents the embedding decision for $\text{skel}(\mu^{①})$ and we again get a binary variable $\text{ord}(\mu^{①})$. Otherwise, this consistency tree contains two or less leaves and can be ignored. Then the choices for the Q-nodes corresponding to $\mu^{①}$ in $T(u^{①})$ and $T(v^{①})$ are independent and we get one binary variable for each of these Q-nodes. We denote these variables by $\text{ord}(\mu^{①}_{u})$ and $\text{ord}(\mu^{①}_{v})$.

We call these variables we get for $G^{①}$ the ①-*PR-ordering variables*. The ②-*PR-ordering* variables are defined analogously. The *PR-ordering variables* are the union of these two variable sets. Let $X^{①}$ be the ⓘ-PR-ordering variables and assume we have a fixed variable assignment $\alpha^{①}$. Then this variable assignment already determines all edge orderings of the common graph, i.e., every embedding of $G^{①}$ realizing the

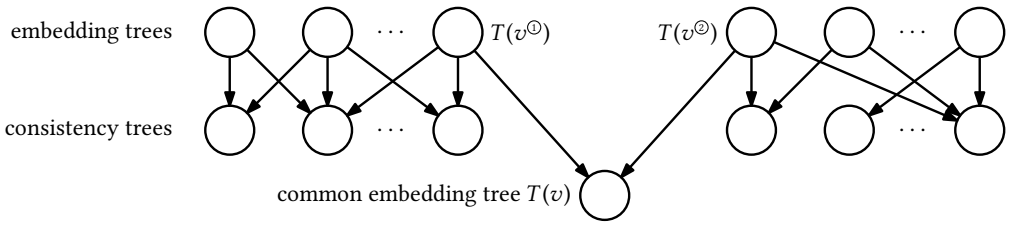


Figure 8.16: The Q-embedding representations of $G^①$ and $G^②$ together with the common embedding tree $T(v)$ of v.

assignment $\alpha^①$ induces the same edge orderings on G. In the following we describe a set of necessary equations and inequalities on the PR-ordering variables that ensure that $G^①$ and $G^②$ induce the same edge orderings on G.

For a common vertex v occurring as $v^①$ and $v^②$ in $G^①$ and $G^②$, respectively, we add a so-called *common embedding tree* $T(v)$ (consisting of a single P-node) as child of the embedding trees $T(v^①)$ and $T(v^②)$ in the Q-embedding representations of $G^①$ and $G^②$; see Figure 8.16. Obviously, this common child ensures that the common edges around v are ordered the same with respect to $G^①$ and $G^②$. Adding the common embedding tree for every common vertex yields the instance $D(G^①, G^②)$.

As the embedding trees (which contain only Q-nodes) are the sources of $D(G^①, G^②)$, normalizing $D(G^①, G^②)$ yields an equivalent instances containing no P-nodes [BR13]; also see Chapter 5. Instances with this property are equivalent to a set of Boolean equations and inequalities, containing one variable for each Q-node in each PQ-tree (and thus includes the PR-ordering variables). We call this set of equations and inequalities the *biconnected PR-ordering constraints*. To obtain the following lemma, it remains to prove the running time.

Lemma 8.24. *Let $G^①$ and $G^②$ be two biconnected graphs with common P-node degree 3 and let α be a variable assignment for the PR-ordering variables. The graphs $G^①$ and $G^②$ admit embeddings that realize α and have consistent edge orderings if and only if α satisfies the biconnected PR-ordering constraints.*

The biconnected PR-ordering constraints have size $O(n)$ and can be computed in $O(n)$ time.

Proof. Let $D(G^①, G^②)$ be the instances of Simultaneous PQ-Ordering as described above. Clearly, $D(G^①, G^②)$ can be constructed in linear time and its size is linear in the size of the input graphs. In general, the equations and inequalities for a given instance of Simultaneous PQ-Ordering can be computed in quadratic time [BR13]. In this specific case, it can be done in linear time for the following reasons.

For every arc in $D(G^①, G^②)$ one needs to compute the normalization of the child (which takes linear time in the size of the parent [BR13]) and a mapping from each inner node of the parent to its representative in the child (which takes again linear time

in the size of the parent [BR13]). When computing the Q-embedding representations for $G^{①}$ and $G^{②}$ from their SPQR-trees (which can be done in linear time), we can make sure that the resulting instances are already normalized and that we already know the mapping from the nodes of the embedding trees to the consistency trees. The remaining arcs in $D(G^{①}, G^{②})$ are arcs from embedding trees to common embedding trees. Thus, for every PQ-tree T in $D(G^{①}, G^{②})$, it suffices to normalize a single outgoing edge, which can be done in time linear in the size of T. □

Allowing Cutvertices

In the following, we extend this result to the case where we allow exclusive cutvertices and simultaneous cutvertices of common degree 3. Let $B_1^{①}, \dots, B_k^{①}$ be the blocks of $G^{①}$ and let $B_1^{②}, \dots, B_\ell^{②}$ be the blocks of $G^{②}$. We say that embeddings of these blocks have *blockwise consistent edge orderings* if for every pair of blocks $B_i^{①}$ and $B_j^{②}$ sharing a vertex v the edges incident to v they share are ordered consistently. To have consistent edge orderings, it is obviously necessary to have blockwise consistent edge orderings.

When composing the embeddings of two blocks that share a cutvertex, the edges of each of the two blocks have to appear consecutively (note that this is no longer true for three or more blocks), which leads to another necessary condition. Let v be an exclusive cutvertex of $G^{①}$. Then v is contained in a single block of $G^{②}$ whose embedding induces an order $O^{②}$ on all common edges incident to v. Let $B_i^{①}$ and $B_j^{①}$ (for $i, j \in \{1, \dots, k\}$ with $i \neq j$) be a pair of blocks containing v and let $O_{i,j}^{②}$ be the order obtained by restricting $O^{②}$ to the common edges in $B_i^{①}$ and $B_j^{①}$. Then the common edges of $B_i^{①}$ must be consecutive in the order $O_{i,j}^{②}$. If this is true for every pair of blocks at every exclusive cutvertex, we say that the embeddings have *pairwise consecutive blocks*.

Lemma 8.25. *Two graphs without simultaneous cutvertices admit embeddings with consistent edge orderings if and only if their blocks admit embeddings that have blockwise consistent edge orderings and pairwise consecutive blocks.*

Proof. Let v be a cutvertex in $G^{①}$ and let $B_1^{①}, \dots, B_k^{①}$ be the blocks containing v. Moreover, let $O^{②}$ be the order of common edges around v given by the unique block of $G^{②}$ containing v. We only have to show that the embeddings of the blocks $B_1^{①}, \dots, B_k^{①}$ can be composed such that they induce the order $O^{②}$ on the common edges. For $k = 2$ this is clear, as we can choose arbitrary outer faces for $B_1^{①}$ and $B_2^{①}$ and combine their embeddings by gluing them together at v. For $k > 2$ it is easy to see that there must be one block, without loss of generality $B_1^{①}$, whose edges appear consecutively in $O^{②}$. The embeddings of all remaining blocks $B_2^{①}, \dots, B_k^{①}$ can be composed by induction such that the edges they contain are ordered the same as in $O^{②}$. Moreover, composing the embedding of $B_1^{①}$ with the resulting embedding of $B_2^{①}, \dots, B_k^{①}$ works the same as the composition of two blocks. □

To extend Lemma 8.24 to the case where we allow exclusive cutvertices, we consider the Q-embedding representations of each block. Ensuring blockwise consistent edge orderings works more or less the same as ensuring consistent edge orderings in the biconnected case. Moreover, we will see how to add additional PQ-trees to ensure pairwise consecutive blocks.

Note that the Q-embedding representations again yield PR-ordering variables. Fixing these variables determines the edge orderings of common edges in each block of $G^{①}$ and in each block of $G^{②}$. If we have no simultaneous cutvertices, every vertex is either not a cutvertex in $G^{①}$ or not a cutvertex in $G^{②}$. Thus, the PR-ordering variables actually determine all edge orderings of the common graph. Thus, although there are new types of embedding choices at the cutvertices in $G^{①}$ and at the cutvertices in $G^{②}$, these choices are already covered by the PR-ordering variables (at least in terms of edge orderings).

Let us formally describe the instance of Simultaneous PQ-Ordering announced above. Let $B^{①}_1, \dots B^{①}_k$ be the blocks of $G^{①}$ and let $B^{②}_1, \dots, B^{②}_\ell$ be the blocks of $G^{②}$. We start with an instance of Simultaneous PQ-Ordering containing the Q-embedding representation of each of these blocks. Let v be a vertex of G that is not a cutvertex. Then v is contained in a single block of $G^{①}$ and in a single block of $G^{②}$, let $v^{①}$ and $v^{②}$ be the occurrences of v in these blocks. As before, there are two embedding trees $T(v^{①})$ and $T(v^{②})$ describing the order of edges around v in $G^{①}$ and $G^{②}$, respectively. As before we can enforce consistent ordering around v by inserting a common embedding tree as a common child of $T(v^{①})$ and $T(v^{②})$.

Let v be a cutvertex of $G^{①}$ (the case that v is a cutvertex of $G^{②}$ is symmetric). Then v occurs in several blocks of $G^{①}$, without loss of generality $B^{①}_1, \dots, B^{①}_r$. We denote the occurrences of v in these blocks by $v^{①}_1, \dots, v^{①}_r$. As v is not a simultaneous cutvertex, it occurs in a single block $B^{②}$ of $G^{②}$. We denote this occurrence by $v^{②}$. The embedding tree $T(v^{②})$ contains a leaf for each of common edge incident to v, thus fixing the order of $T(v^{②})$ already fixes the order of all common edges around v. To ensure consistency, we have to enforce that conditions of Lemma 8.25, i.e., blockwise consistent edge orderings and pairwise consecutive blocks.

Ensuring blockwise consistent edge orderings is equivalent to enforcing that the common edges incident to v in $B^{①}_i$ (for $i = 1, \dots, k$) are ordered the same with respect to the Q-embedding representations of $B^{①}_i$ and $B^{②}$. This can be done by inserting a new child consisting of a single P-node as common child of $T(v^{②})$ and $T(v^{①}_i)$. We call this tree the *blockwise common embedding tree*.

To ensure pairwise consecutive blocks, consider the two blocks $B^{①}_i$ and $B^{①}_j$. We create a PQ-tree $T_{i,j}(v^{①})$ that has a leaf for each common edge incident to $v^{①}$ that belongs to one of the two blocks $B^{①}_i$ or $B^{①}_j$. The structure of $T_{i,j}(v^{①})$ is chosen such that all common edges in $B^{①}_i$ are consecutive; see Figure 8.17. We call $T_{i,j}(v^{①})$ *pairwise*

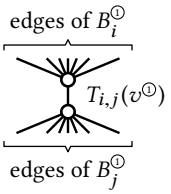


Figure 8.17: A pairwise consecutivity tree $T_{i,j}(v^{①})$.

consecutivity trees and add it as a child of the embedding tree $T(v^{②})$, which has a leaf for every common edge incident to v.

We denote the resulting instance of Simultaneous PQ-Ordering by $D(G^{①}, G^{②})$. As before, all sources in $D(G^{①}, G^{②})$ contain only Q-nodes. Thus, normalizing $D(G^{①}, G^{②})$ leads to an instance containing only Q-nodes, which is again equivalent to a set of equations and inequalities. We call this set the *PR-ordering constraints*.

So far, the PR-ordering constraints only ensure blockwise consistent edge orderings and pairwise consecutive blocks if every cutvertex is an exclusive cutvertex. Recall that there is no need for handling union cutvertices; see Theorem 8.1. Assume we allow simultaneous cutvertices of common degree 3 and let v be such a cutvertex. Then there are two possibilities. If v does not separate the three common edges incident to v in one of the graphs $G^{①}$ (for $i \in \{1,2\}$), then the PR-ordering variables of $G^{①}$ also determine the common edge ordering around v and thus this simultaneous cutvertex actually behaves like an exclusive cutvertex. Otherwise, the common edges incident to v are separated by v in $G^{①}$ and in $G^{②}$. Thus, changing the edge ordering of the common edges at v in an embedding of $G^{①}$ has no effect on any other edge ordering. As the same holds for $G^{②}$, we actually choose an arbitrary edge ordering for the common edges incident to v, independent from all other edge orderings.

Thus, we do not need to add additional constraints for the case that we allow simultaneous cutvertices of common degree 3. To obtain the following lemma, it remains to prove the running time.

Lemma 8.26. *Let $G^{①}$ and $G^{②}$ have common P-node degree 3 and simultaneous cutvertices of common degree at most 3. Let α be a variable assignment for the PR-ordering variables. The graphs $G^{①}$ and $G^{②}$ admit embeddings that realize α and have consistent edge orderings if and only if α satisfies the PR-ordering constraints.*

The PR-ordering constraints have size $O(n^2)$ and can be computed in $O(n^2)$ time.

Proof. The size of the PR-ordering constraints is linear in the size of the instance $D(G^{①}, G^{②})$. Clearly, the Q-embedding representations of the blocks have overall linear size. The common embedding tree for a vertex v has size $O(\deg(v))$. Similarly, the blockwise common embedding trees of a vertex v have total size $O(\deg(v))$. However, if $G^{①}$ has a linear number of blocks incident to a vertex v, then we get a quadratic number of pairwise consecutivity trees.

To get the PR-ordering constraints from the instance $D(G^{①}, G^{②})$ we have to compute

for each arc in $D(G^{①}, G^{②})$ the normalization and the mapping from the Q-nodes of the source to their representatives in the target. As in the proof of Lemma 8.24, the arcs of $D(G^{①}, G^{②})$ belonging to the Q-embedding representation can be computed in linear time. All remaining arcs have an embedding tree as source. An embedding tree of a vertex v has only $O(\deg(v))$ common embedding trees or blockwise common embedding trees as children. Processing all arcs to these children takes $O(\deg(v)^2)$ time and thus overall $O(n^2)$ time.

It remains to deal with arcs of the following type. The source is the embedding tree $T(v^{②})$ and the target is the pairwise consecutivity tree $T_{i,j}(v^{①})$ for a pair of blocks $B_i^{①} \neq B_j^{①}$. Normalizing such an arc would usually take $O(\deg(v))$ time, which has to be done for $O(\deg(v)^2)$ pairwise consecutivity trees, resulting in the running time $O(\deg(v)^3)$. To improve this, we can make use of the fact that the subtree of $T_{i,j}(v^{①})$ (after the normalization) that represents only the edges in $B_i^{①}$ has always the same structure, independent of the other block $B_i^{②}$.

We first compute for each block $B_i^{①}$ the reduction of $T(v^{②})$ with the leaves belonging to $B_i^{①}$, which takes $O(\deg(v))$ time for each of the $O(\deg(v))$ blocks. The resulting tree contains a single node η_i separating the leaves belonging to $B_i^{①}$ from all other leaves. We project this tree to the leaves belonging to $B_i^{①}$ to obtain the tree $T_i(v^{①})$ with root η_i. Computing these trees $T_i(v^{①})$ together with the mapping from the nodes in $T(v^{②})$ to their representatives in $T_i(v^{①})$ takes $O(\deg(v))$ time for each block and thus overall $O(\deg(v)^2)$ time.

The desired (normalized) pairwise consecutivity tree $T_{i,j}(v^{①})$ can be obtained by identifying the roots η_i and η_j of the trees $T_i(v^{①})$ and $T_j(v^{①})$ with each other. Extending the mapping to the resulting tree $T_{i,j}(v^{①})$ can be easily done in linear time in the size of $T_{i,j}(v^{①})$. Hence, for the whole instance, we get the running time $O(n^2)$. □

Cutvertex-Ordering Variables

Let v be an exclusive cutvertex of $G^{①}$ and let $B_1^{①}, \dots, B_k^{①}$ be the blocks incident to v that include common edges incident to v. As mentioned before, the choice of how these blocks are ordered around v and how they are nested into each other is determined by the PR-ordering variables of $G^{②}$. Consider a pair of blocks $B_i^{①}$ and $B_j^{①}$. As the choice of how $B_i^{①}$ and $B_j^{①}$ are embedded into each other at v may also have an effect on some relative positions, we would like to have more direct access to this information (not only via the PR-ordering variables of $G^{②}$).

Let $B^{①} \in \{B_1^{①}, \dots, B_k^{①}\}$ be a block of $G^{①}$ that contains the common edge e incident to v and let e_1 and e_2 be two common edges incident to v that are contained in one block distinct from $B^{①}$. We create a variable $\mathrm{ord}(e_1, e_2, B^{①})$ to represent the binary decision of ordering the edges e_1, e_2, and e in this order or in its reversed order. Note that this is independent from the choice of the edge e of $B^{①}$ (the blocks are pairwise

consecutive in every embedding). We create such a variable for every such triple e_1, e_2, and $B^{①}$ and call them the *exclusive cutvertex-ordering variables*.

To make sure that the exclusive cutvertex-ordering variables are consistent with the PR-ordering variables, it suffices to slightly change the above instance $D(G^{①}, G^{②})$ of Simultaneous PQ-Ordering. Let $\text{ord}(e_1, e_2, B^{①})$ be an exclusive cutvertex-ordering variable for the cutvertex v. Let e be a common edge in $B^{①}$ incident to v and let $v^{②}$ be the occurrence of v in $G^{②}$. Then $D(G^{①}, G^{②})$ contains the embedding tree $T(v^{②})$ that has a leaf for every common edge incident to v. This includes e_1, e_2, and e. We create a new PQ-tree $T(e_1, e_2, e)$ with three leaves corresponding to e_1, e_2, and e and add it as child of $T(v^{②})$. In every solution of the resulting instance of Simultaneous PQ-Ordering, the value of $\text{ord}(e_1, e_2, B^{①})$ then simply corresponds to the orientation chosen for PQ-tree $T(e_1, e_2, e)$.

Adding this tree for every exclusive cutvertex-ordering variable establishes the desired connection between these variables and the PR-ordering variables. We call the constraints we get from the resulting instance of Simultaneous PQ-Ordering in addition to the PR-ordering constraints the *cutvertex-ordering constraints*.

Let v be a simultaneous cutvertex of common degree 3 such that the common edges incident to v are separated by v in $G^{①}$ and in $G^{②}$. Recall that this is the unique case, where PR-ordering variables do not determine the order of the common edges around v. Let e_1, e_2, and e_3 be the common edges incident to v. To make sure that assigning values to all variables actually determines all edge-orderings, we add the variable $\text{ord}(e_1, e_2, e_3)$ associated with the order of these three edges. Recall that changing this order in $G^{①}$ or $G^{②}$ has no effect on any other edge ordering. Hence, there is no need to add further constraints. If two of the edges, without loss of generality e_1 and e_2, belong to the same block of $G^{①}$ and e_3 belongs to another block $B^{①}$, we denote $\text{ord}(e_1, e_2, e_3)$ also by $\text{ord}(e_1, e_2, B^{①})$ to obtain consistency with the naming of the exclusive cutvertex-ordering variables.

We call these variables together with the exclusive cutvertex-ordering variables the *cutvertex-ordering variables*. The PR-ordering variables together with the cutvertex-ordering variables are simply called *ordering variables*. Moreover, the PR-ordering constraints together with the cutvertex-ordering constraints are called *ordering constraints*. To extend Lemma 8.26 to incorporate the cutvertex-ordering variables, it remains to show that the cutvertex-ordering constraints have $O(n^3)$ size and can be computed in $O(n^3)$ time.

Lemma 8.27. *Let $G^{①}$ and $G^{②}$ have common P-node degree 3 and simultaneous cutvertices of common degree at most 3. Let α be a variable assignment for the ordering variables. The graphs $G^{①}$ and $G^{②}$ admit embeddings that realize α and have consistent edge orderings if and only if α satisfies the ordering constraints.*

The ordering constraints have size $O(n^3)$ and can be computed in $O(n^3)$ time.

Proof. The size of the cutvertex-ordering constraints is clearly linear in the number of

cutvertex-ordering variables. For each cutvertex v, the number of cutvertex-ordering variables is clearly in $O(\deg(v)^3)$. Thus, it remains to show how to get the cutvertex-ordering constraints from the resulting instance $D(G^①, G^②)$ of Simultaneous PQ-Ordering.

Let v be a cutvertex in $G^①$ and let $v^②$ be the occurrence of v in $G^②$. For every cutvertex-ordering variable of v we have three common edges e_1, e_2, and e and we need to find the node η of the embedding tree $T(v^②)$ that separates the leaves corresponding to e_1, e_2, and e from each other. When rooting $T(v^②)$ at e, this node η is the lowest common ancestor of e_1 and e_2. Thus, after $O(\deg(v))$ preprocessing time, we can get the cutvertex-ordering constraint for every cutvertex-ordering variable that includes e in constant time per variable [HT84]. We have to spend this $O(\deg(v))$ preprocessing time at most once for each common edge incident to v, yielding a total preprocessing time of $O(\deg(v)^2)$. As we have $O(\deg(v)^3)$ cutvertex-ordering variables for v, the running time is dominated by the constant time LCA-queries, which yield the running time $O(\deg(v)^3)$. For the whole instance, this gives the claimed $O(n^3)$ running time. □

8.4.3 Common-Face Constraints

Recall from Section 8.2.4 that we can assume that there are no bridges that are block-local and exclusive one-attached if we in return solve Sefe with block-local common-face constraints. In this section, we show how to handle these additional constraints. To this end, we show that satisfying block-local common-face constraints in a given instance of Sefe is equivalent to satisfying a set of equations and inequalities. The union of these constraints with the constraints from the previous section thus enforce that the embeddings of each common connected component are consistent and satisfy given block-local common-face constraints.

Let B be a block of the common graph. Let μ be an R-node of B. Then we introduce the binary variable $\text{ord}(\mu)$ where $\text{ord}(\mu) = 0$ indicates that $\text{skel}(\mu)$ is embedded according to its reference embedding and $\text{ord}(\mu) = 1$ indicates that $\text{skel}(\mu)$ is flipped. In case μ is a P-node of B, we can assume by Theorem 8.2 that the union-link graph of μ is connected. Recall that this implies that the embedding of $\text{skel}(\mu)$ is fixed up to a flip (Lemma 8.2). Thus, we also get a reference embedding for $\text{skel}(\mu)$ and can describe the embedding choice for $\text{skel}(\mu)$ with a binary variable $\text{ord}(\mu)$. We call these variables the *common PR-node variables*.

It follows directly from Section 8.3.3 that common-face constraints for B are equivalent to a set of equations and inequalities on the variables $\text{ord}(\mu)$ (we basically get the consistency and union-bridge constraints from Step 2 and Step 4).

Note that the constraints from Section 8.4.2 enforcing consistent edge orderings do not contain common PR-node variables. They only contain PR-ordering variables determining the embeddings of $G^①$ and $G^②$ (Lemma 8.26). As fixing the PR-node variables for $G^①$ fixes the embedding of G, it also fixes the values for all common

PR-node variables. It remains to show that this dependency of the common PR-node variables from the PR-ordering variables in $G^{①}$ can be expressed using a set of equations and inequalities.

To this end, consider a common vertex v in the common block B and let $B^{①}$ be the block of $G^{①}$ containing B. Let $T(v)$ be the embedding tree of v in B, i.e., the PQ-tree describing the possible edge orderings of common edges incident to v. Note that each inner node of $T(v)$ is actually a Q-node and that $T(v)$ has one inner node for each P- or R-node whose embedding affects the edge ordering around v. Let $v^{①}$ be the occurrence of v in $B^{①}$ and let $T(v^{①})$ be the embedding tree of $v^{①}$ in $B^{①}$ projected to the common edges incident to v. Then $T(v^{①})$ describes all orders of common edges incident to v that can be induced by an embedding of $B^{①}$.

Clearly, $T(v^{①})$ is more restrictive than $T(v)$ in the sense that every order represented by $T(v^{①})$ is also represented by $T(v)$. Thus, $T(v^{①})$ is a reduction of $T(v)$. It is not hard to see that every Q-node in $T(v)$ has a unique Q-node in $T(v^{①})$ (called its *representative*; see Chapter 5) that determines its flip. Thus, for every P- or R-node μ of G, we find at least one vertex v such that $\mathrm{ord}(\mu)$ corresponds to the flip of a Q-node in $T(v)$, which corresponds to a flip of a Q-node in $T(v^{①})$, which corresponds to a PR-ordering variable $\mathrm{ord}(\mu^{①})$ (or $\mathrm{ord}(\mu^{①}_{v})$) of $G^{①}$. Thus, $\mathrm{ord}(\mu) = \mathrm{ord}(\mu^{①})$ or $\mathrm{ord}(\mu) \neq \mathrm{ord}(\mu^{①})$ gives us the desired connection between the common PR-node variables and the PR-ordering variables.

We call the set of all equations and inequalities described in this section the *common-face constraints*. With the results form Section 8.3.3 (and with standard PQ-tree operations), we can compute the common-face constraints in linear time. This yields the following lemma where n is the total input size, i.e., the size of the two graphs plus the size of the common-face constraints.

Lemma 8.28. *Let $(G^{①}, G^{②})$ be an instance of Sefe with common-face constraints and let α be a variable assignment for the PR-ordering and the common PR-node variables. Every embedding of $G^{①}$ realizing α satisfies the common-face constraints if and only if α satisfies the common-face constraints.*

The common-face constraints have $O(n)$ size and can be computed in $O(n)$ time.

8.4.4 Consistent Relative Positions

Let H and H' be two connected components of the common graph G. To represent the relative position $\mathrm{pos}_{H'}(H)$ of H with respect to H', we use the relative positions $\mathrm{pos}_{C}(H)$ of H with respect to the cycles C in an extended cycle basis of H'. With $\mathrm{pos}_{C}(H) = 0$ and $\mathrm{pos}_{C}(H) = 1$, we associate the cases that H lies to the right of C and to the left of C, respectively. We call these variables the *component position variables*. Note that fixing all position variables determines all relative positions of common connected components with respect to each other (Theorem 8.8). Thus, fixing

the PR-ordering variables (which also fixes the cutvertex-ordering variables) and the position variables completely determines the embedding of the common graph G.

In this section, we give a set of necessary equations and inequalities on the position variables of two graphs $G^{①}$ and $G^{②}$ that enforce consistent relative positions on their common graph G. As fixing the PR-ordering variables may also determine some position variables, we also have to make sure that these two types of variables are consistent with each other.

Let $G^{①}$ be a connected planar graph containing G, let C be a cycle in G (a cycle from the extended cycle basis), and let H be a connected component of G not containing C. Depending on how C and H are located in $G^{①}$, different embedding choices of $G^{①}$ determine the relative position $\mathrm{pos}_C(H)$; see Chapter 7. We quickly list these embedding choices here and describe the constraints arising from them in the following sections.

Let $\mu^{①}$ be an R-node of $G^{①}$ such that C induces a cycle κ in $\mathrm{skel}(\mu^{①})$. If $\mathrm{skel}(\mu^{①})$ has a virtual edge ε that is not part of κ such that the expansion graph $\mathrm{expan}(\varepsilon)$ includes a vertex of H, then the embedding of $\mathrm{skel}(\mu^{①})$ determines the relative position $\mathrm{pos}_C(H)$. We say that $\mathrm{pos}_C(H)$ is *determined by the R-node* $\mu^{①}$.

Let $\mu^{①}$ be a P-node of $G^{①}$ such that C induces a cycle κ in $\mathrm{skel}(\mu^{①})$. Then κ consists of two virtual edges ε_1 and ε_2. The relative position $\mathrm{pos}_C(H)$ is determined by the embedding of $\mathrm{skel}(\mu^{①})$ if H is contained in the expansion graph of a virtual edge ε with $\varepsilon \neq \varepsilon_i$ for $i \in \{1,2\}$. We say that $\mathrm{pos}_C(H)$ is *determined by the P-node* $\mu^{①}$.

If C and H belong to the same block $B^{①}$ of $G^{①}$, then $\mathrm{pos}_C(H)$ is either determined by an R-node or by a P-node of $B^{①}$; see Chapter 7. Otherwise, let v be the cutvertex in $G^{①}$ that separates C from H and belongs to the block of C. If v is not contained in C, then we introduce the variable $\mathrm{pos}_C(v)$ corresponding to the decision of embedding v to the right or to the left of C. We call such a variable the *cutvertex position variables*. Clearly, H and v lie on the same side in each embedding of $G^{①}$. Moreover, the relative position of v with respect to C is determined by an R-node or by a P-node $\mu^{①}$ (the belong to the same block of $G^{①}$). In this case, we also say that both variables, $\mathrm{pos}_C(v)$ and $\mathrm{pos}_C(H)$, are determined by the R-node or P-node $\mu^{①}$. Moreover, we also say that $\mathrm{pos}_C(H)$ is *determined by* $\mathrm{pos}_C(v)$.

If the cutvertex v is contained in C, then the relative position $\mathrm{pos}_C(H)$ is determined by the embedding choices made at the cutvertex. We distinguish two cases. Let $S^{①}$ be the split component with respect to the cutvertex v that contains H. If $S^{①}$ includes a common edge incident to v, we say that $\mathrm{pos}_C(H)$ is *determined by the common cutvertex* v. Otherwise, we say that $\mathrm{pos}_C(H)$ is *determined by the exclusive cutvertex* v (note that v might still be a cutvertex of the common graph in this case). These two cases are in so far different as changing $\mathrm{pos}_C(H)$ affects the edge ordering of the common graph in the former case, whereas it does not in the latter case.

The component position variables together with the cutvertex position variables are

simply called *position variables* In the following sections, we describe for each of the four cases different constraints in the form of equations and inequalities on position variables, PR-ordering variables, and cutvertex-ordering variables.

Relative Positions Determined by R-Nodes

We start with the simplest case, where $\text{pos}_C(H)$ is determined by an R-node $\mu^{①}$ of $G^{①}$. If $\text{pos}_C(H)$ is also determined by a cutvertex position variable $\text{pos}_C(v)$, we simply set $\text{pos}_C(H) = \text{pos}_C(v)$ to make sure that the cutvertex v and the component H are on the same side of C.

Otherwise, C induces a cycle κ in $\text{skel}(\mu^{①})$ and H shares a vertex with the expansion graph of a virtual edge ε not belonging to κ. The PR-ordering variable $\text{ord}(\mu^{①})$ determines whether $\text{skel}(\mu^{①})$ is embedded according to its reference embedding ($\text{ord}(\mu^{①}) = 0$) or whether it is flipped ($\text{ord}(\mu^{①}) = 1$).

Assume ε lies to the right of κ in the reference embedding of $\text{skel}(\mu^{①})$. Then $\text{ord}(\mu^{①}) = 0$ implies that ε lies to the right of κ, which implies that H lies to the right of C, i.e., $\text{pos}_C(H) = 0$. Moreover, flipping $\text{skel}(\mu^{①})$ brings ε to the left of κ. Thus, $\text{ord}(\mu^{①}) = 1$ implies $\text{pos}_C(H) = 1$, which yields $\text{ord}(\mu^{①}) = \text{pos}_C(H)$ as necessary condition. If ε lies to the left of κ in the reference embedding, we obtain $\text{ord}(\mu^{①}) \neq \text{pos}_C(H)$ instead.

Analogously, we set $\text{ord}(\mu^{①}) = \text{pos}_C(v)$ or $\text{ord}(\mu^{①}) \neq \text{pos}_C(v)$ for every cutvertex position variable $\text{pos}_C(v)$ determined by $\mu^{①}$.

Sufficiency. We call the constraints defined in this section the *R-node constraints*. The following lemma states the more or less obvious necessity and sufficiency of the R-node constraints.

Lemma 8.29. *Let $\mu^{①}$ be an R-node of $G^{①}$ and let X be the set of position variables determined by $\mu^{①}$ together with the PR-ordering variable* $\text{ord}(\mu^{①})$*. A variable assignment α of X can be realized by an embedding of $G^{①}$ if and only if α satisfies the R-node constraints.*

Relative Positions Determined by P-Nodes

Let $\mu^{①}$ be a P-node of $G^{①}$ with poles u and v and let $\varepsilon_1, \ldots, \varepsilon_k$ be the virtual edges of $\text{skel}(\mu^{①})$. Let C be a cycle in G, that induces a cycle κ in $\text{skel}(\mu^{①})$. Without loss of generality, κ consists of the virtual edges ε_1 and ε_2. Let H be a common connected component whose relative position with respect to C is determined by $\mu^{①}$.

As for R-nodes, we first consider the case that a relative position $\text{pos}_C(H)$ is determined by a cutvertex position variable $\text{pos}_C(v)$. In this case we simply set $\text{pos}_C(H) = \text{pos}_C(v)$. In the following we assume that $\text{pos}_C(H)$ is determined by the

P-node $\mu^{①}$ but not by a cutvertex position variable. All cutvertex position variables $\text{pos}_C(v)$ are handled analogously.

As $\text{pos}_C(H)$ is determined by $\mu^{①}$, the common connected component H is contained in a virtual edge $\varepsilon \in \{\varepsilon_3, \dots, \varepsilon_k\}$ not belonging to κ. Note that embedding ε to the right or to the left of κ determines not only the relative position of H but the position of every connected component that is contained in the expansion graph of ε. We make sure that these relative positions fit to each other by introducing a new variable $\text{pos}_\kappa(\varepsilon)$ with the interpretation that $\text{pos}_\kappa(\varepsilon) = 0$ if and only if ε is embedded to the right of κ. Clearly, $\text{pos}_\kappa(\varepsilon) = \text{pos}_C(H)$ is a necessary condition for every connected component H contained in the expansion graph of ε.

If there is another common cycle C' inducing the same cycle κ in $\text{skel}(\mu)$, we use the same variable $\text{pos}_\kappa(\varepsilon)$ to determine on which side of κ the virtual edge ε is embedded. If C' induces the same cycle oriented in the opposite direction, we use the negation of $\text{pos}_\kappa(\varepsilon)$ instead.

The above constraints are not sufficient for two reasons. First, changing the embedding of $\text{skel}(\mu^{①})$ may change edge orderings in the common graph. In this case, there are PR-ordering variables partially determining the embedding of $\text{skel}(\mu^{①})$ and we have to make sure that their values and the values of the position variables determined by $\mu^{①}$ fit to each other. Second, if different common cycles induce different cycles in $\text{skel}(\mu^{①})$, then not every combination of relative positions with respect to these cycles can actually be achieved by embedding the skeleton.

Connection to Ordering Variables. As mentioned before, we have to add additional constraints to ensure consistency between the position variables and the ordering variables. Let κ be the cycle induced by C in $\text{skel}(\mu^{①})$ and assume without loss of generality that κ consists of the virtual edges ε_1 and ε_2. Embedding another virtual edge ε to the right or to the left of κ changes the edge ordering at the poles u or v if and only if the expansion graph of ε includes common edges incident to u or v, respectively. In this case, we have a PR-ordering variable determining this embedding choice and we can make sure that the edge ordering and the relative positions fit to each other using an equation or an inequality.

To make this more precise, we define the following five types of virtual edges. Each virtual edge $\varepsilon \in \{\varepsilon_1, \dots, \varepsilon_k\}$ is of exactly one of the following five types.

Type 1. $\text{expan}(\varepsilon)$ includes a common path from u to v.

Type 2. $\text{expan}(\varepsilon)$ has common edges incident to u and to v but is not of Type 1.

Type 3. $\text{expan}(\varepsilon)$ has a common edge incident to u but none incident to v.

Type 4. $\text{expan}(\varepsilon)$ has a common edge incident to u but none incident to v.

Type 5. $\text{expan}(\varepsilon)$ has no common edges incident to u or to v.

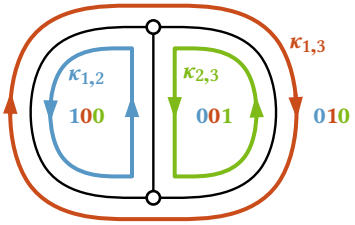


Figure 8.18: Three virtual edges of Type 1 in a P-node with the three different virtual cycles $\kappa_{1,2}$, $\kappa_{1,3}$, and $\kappa_{2,3}$. For each face, the variable assignment corresponding to this face is given.

As the cycle κ consists of ε_1 and ε_2, they must both be of Type 1. Choosing the relative position of ε with respect to κ affects the edge ordering at u or at v if and only if ε is not of Type 5. If ε is of Type 1 or Type 2, then there is a PR-ordering variable $\text{ord}(\mu^{①})$ determining the order of the edges $\varepsilon_1, \varepsilon_2, \varepsilon$. If the ordering corresponding to $\text{ord}(\mu^{①}) = 0$ has ε to the right of κ, we set $\text{ord}(\mu^{①}) = \text{pos}_{\kappa}(\varepsilon)$. Otherwise, we set $\text{ord}(\mu^{①}) \neq \text{pos}_{\kappa}(\varepsilon)$.

If ε is of Type 3, the edge ordering of $\varepsilon_1, \varepsilon_2, \varepsilon$ is determined by the PR-ordering variable $\text{ord}(\mu_s^{①})$. As before, we get either the equation $\text{ord}(\mu_s^{①}) = \text{pos}_{\kappa}(\varepsilon)$ or the inequality $\text{ord}(\mu_s^{①}) \neq \text{pos}_{\kappa}(\varepsilon)$. The case that ε is of Type 4 is analogous, except that we have the PR-ordering variable $\text{ord}(\mu_t^{①})$ instead of $\text{ord}(\mu_s^{①})$.

Multiple Cycles. If there are common cycles inducing different cycles in the P-node $\mu^{①}$, then at least three virtual edges must be of Type 1 (i.e., their expansion graph includes a common path between the poles u and v). As we assume that $\mu^{①}$ has common P-node degree 3, three virtual edges are of Type 1 and all remaining virtual edges are of Type 5. Let ε_1, ε_2, and ε_3 be the virtual edges of Type 1 and let ε be another virtual edge of $\text{skel}(\mu^{①})$. Denote the cycle consisting of ε_i and ε_j by $\kappa_{i,j}$ ($i, j \in \{1, 2, 3\}$ and $i < j$). To simplify the notation, we use $\text{pos}_{i,j}(\varepsilon)$ as short form for the relative position $\text{pos}_{\kappa_{i,j}}(\varepsilon)$.

Let $\varepsilon \in \{\varepsilon_4, \ldots, \varepsilon_k\}$ be another virtual edge of $\text{skel}(\mu^{①})$. Then we are interested in the three position variables $\text{pos}_{1,2}(\varepsilon)$, $\text{pos}_{1,3}(\varepsilon)$, and $\text{pos}_{2,3}(\varepsilon)$, which are not independent from each other. Moreover, which combinations of relative positions can actually be realized depends on the ordering of ε_1, ε_2, and ε_3. This ordering is determined by the PR-ordering variable $\text{ord}(\mu^{①})$. In the remainder of this section, we first figure out which combinations of values for $\text{ord}(\mu^{①})$, $\text{pos}_{1,2}(\varepsilon)$, $\text{pos}_{1,3}(\varepsilon)$, and $\text{pos}_{2,3}(\varepsilon)$ are actually possible and then show that restricting the variables to these combinations is equivalent to a set of equations and inequalities.

When fixing the order variable $\text{ord}(\mu^{①})$ (without loss of generality to 0), we get the situation shown in Figure 8.18. From the set of eight combinations for the three position variables, only the three combinations 100, 010 and 001 are possible. When changing the order of the edges (setting $\text{ord}(\mu^{①})$ to 1), every bit is reversed. Thus, for the tuple $(\text{ord}(\mu^{①}), \text{pos}_{1,2}(\varepsilon), \text{pos}_{1,3}(\varepsilon), \text{pos}_{2,3}(\varepsilon))$ we get the possibilities 0100, 0010, 0001 and their complements 1011, 1101, 1110. A restriction equivalent to this is called

P-node 4-constraint (where equivalent means that an arbitrary subset of variables may be negated).

We note that, in the short version of this paper [BKR13a], we missed the fact that the combinations 1000 and 0111 are not possible. This lead to the wrong assumption that a combination is feasible if and only if there is an odd number of 1s, which can be expressed as a linear equation over $\mathbb{F}_2$. As a matter of fact, the P-node 4-constraint allows six different combinations, which is not a power of two and can thus not be the solution space of a linear equation over $\mathbb{F}_2$. Thus, a P-node 4-constraint is in particular not equivalent to a set of equations or inequalities. We resolve this issue with the following lemma in conjunction with the new results from Section 8.2.2.

Lemma 8.30. *If two variables of a P-node 4-constraint are known to be equal or unequal, the P-node 4-constraint is equivalent to a set of equations and inequalities.*

Proof. We basically have the three possibilities 0100, 0010, and 0001 and their complements for the variables *abcd*. No pair of variables is equal in all three possibilities and no pair of variables in unequal in all three possibilities. Thus requiring equality or inequality for one of the pairs eliminates exactly one or two of these three possibilities. If exactly one possibility and its complement remains, this is obviously equivalent to a set of equations and inequalities.

If 0100 and 0010 (and their complements) remain, this is equivalent to $a = d$ and $b \neq c$. If 0100 and 0001 remain, this is equivalent to $a = c$ and $b \neq d$. Finally, if 0010 and 0001 remain, this is equivalent to $a = b$ and $c \neq d$. □

In the following, we show that for every P-node 4-constraint, we always find an equation or inequality between a pair of variables, turning all P-node 4-constraints into a set of equations and inequalities.

Consider the union graph $G^\cup$. If the poles $\{u, v\}$ are a separating pair in $G^\cup$, then each split component is the union of the expansion graphs of several virtual edges of skel$(\mu^{①})$. As the expansion graphs of ε_1, ε_2, and ε_3 have common uv-paths, we can assume that they are not separated (Theorem 8.3). Moreover, having common P-node degree 3 implies that none of the other expansion graphs has a common edge incident to u or to v (they are all of Type 5). Thus, again by Theorem 8.3, we can assume that $\{u, v\}$ is not a separating pair in $G^\cup$.

It follows that there must be a path π in $G^\cup$ that connects a vertex of expan(ε) with a vertex of (without loss of generality) expan(ε_1) that does not pass through u or v or vertices of the expansion graphs of ε_2 and ε_3. It follows that the relative position of ε with respect to $\kappa_{2,3}$ must be the same as the relative position of any internal vertex of expan(ε_1) with respect to $\kappa_{2,3}$. As this relative position is determined by the order of ε_1, ε_2, and ε_3, we obtain either $\text{ord}(\mu^{①}) = \text{pos}_{2,3}(\varepsilon)$ or $\text{ord}(\mu^{①}) \neq \text{pos}_{2,3}(\varepsilon)$.

Sufficiency. We call the constraints defined in this section the *P-node constraints*. The following lemma follows directly from the previous considerations.

Lemma 8.31. *Let $\mu^{①}$ be a P-node of $G^{①}$ and let X be the set of position variables determined by $\mu^{①}$ together with the PR-ordering variables of $\mu^{①}$. A variable assignment α of X can be realized by an embedding of $G^{①}$ if and only if α satisfies the P-node constraints.*

Relative Positions Determined by Common Cutvertices

Recall that the relative position $\mathrm{pos}_C(H)$ is determined by a common cutvertex v of $G^{①}$ if C contains v and H lies in a split component $S^{①}$ (with respect to v) different from the split component containing C such that $S^{①}$ has a common edge incident to v.

First note that the whole split component $S^{①}$ has to be embedded on one side of C. Thus, for every common connected component in $S^{①}$, we would get the same set of constraints. To reduce the number of constraints, we introduce the variable $\mathrm{pos}_C(S^{①})$ representing the decision of embedding $S^{①}$ either to the right or to the left of C. Clearly, $\mathrm{pos}_C(H) = \mathrm{pos}_C(S^{①})$ for every common connected component H in $S^{①}$ is a necessary condition.

Note that this condition is very similar to the first type of constraints we required for P-nodes (connected components in the expansion graph of the same virtual edge have the same relative positions). As for the P-nodes, we have to address two potential issues. First, embedding the split component $S^{①}$ to one side or another of C changes the edge ordering around the cutvertex v. Second, if there are multiple cycles through v, then the relative positions of $S^{①}$ with respect to all these cycles must be consistent.

Connection to Ordering Variables. Let $B^{①}$ be the block of $S^{①}$ containing v and let e_1 and e_2 be the two edges of C incident to v. Moreover, let e be a common edge of $B^{①}$ incident to v. Recall that the cyclic order of e_1, e_2, and e is described by the cutvertex-ordering variable $\mathrm{ord}(e_1, e_2, B^{①})$.

Assume without loss of generality that e_1 is oriented towards v and e_2 is oriented away from v (in the cycle C). Then the (clockwise) cyclic order e_1, e, e_2 forces the block $B^{①}$, and thus the whole split component $S^{①}$, to lie left of C. The opposite cyclic order forces $S^{①}$ to the right of C. Thus, depending on the orientation of C, we either get $\mathrm{ord}(e_1, e_2, B^{①}) = \mathrm{pos}_C(S^{①})$ or $\mathrm{ord}(e_1, e_2, B^{①}) \neq \mathrm{pos}_C(S^{①})$ as necessary conditions.

Multiple Cycles. Assume multiple common cycles $C_1, \dots, C_k$ contain the cutvertex v and assume that these cycles are already embedded. We have to make sure that every assignment to the variables $\mathrm{pos}_{C_i}(S^{①})$ for $i = 1, \dots, k$ actually corresponds to a face of $C_1 \cup \cdots \cup C_k$ that is incident to v.

We cannot directly express this requirement as a set of equations and inequalities on the position variables. However, if we assume that a given variable assignment for the cutvertex-ordering variables of v can be realized by an embedding of $G^{①}$ (which is ensured by the constraints from Section 8.4.2), then the above constraints establishing the connection between the cutvertex-ordering variables and the position variables make sure that the corresponding values for the position variables are also realized.

Sufficiency. We call the constraints from this section the *common cutvertex constraints*. Let α be a variable assignment for the cutvertex-ordering variables of a cutvertex v. We say that α is *order realizable*, if $G^{①}$ admits an embedding realizing α. We obtain the following lemma.

Lemma 8.32. *Let X be the set of position variables that are determined by the common cutvertex v in $G^{①}$ and let Y be the cutvertex-ordering variables of v. A variable assignment α of $X \cup Y$ can be realized by an embedding of $G^{①}$ if and only if α satisfies the common cutvertex constraints and $\alpha|_Y$ is order realizable.*

Relative Positions Determined by Exclusive Cutvertices

As in the previous section, let $S^{①}$ be the split component with respect to the cutvertex v that contains the connected component H. As before, every common connected component of $S^{①}$ has to be embedded on the same side of C. However, in this case, we need slightly stronger constraints.

Let H_v be the connected component of the common graph that includes the cutvertex v. Let further $B_1^{\cup}, \ldots, B_k^{\cup}$ be the union bridges of H_v (note that this is the first time, where the second graph $G^{②}$ comes into play). As the union bridge $B_i^{\cup}$ (for $i = 1, \ldots, k$) has to be completely embedded into a single face of H_v, every common connected component in $B_i^{\cup}$ lies on the same side of C. As before for the split components, we represent the decision of putting $B_i^{\cup}$ to the right or to the left of C using the variable $\mathrm{pos}_C(B_i^{\cup})$. Then the constraint $\mathrm{pos}_C(B_i^{\cup}) = \mathrm{pos}_C(H)$ for every common connected component H in $B_i^{\cup}$ is clearly necessary. Note that the resulting constraints are strictly stronger than setting $\mathrm{pos}_C(S^{①}) = \mathrm{pos}_C(H)$ for every common connected component H in $S^{①}$, as $S^{①}$ is contained in a single union bridge.

Recall that (in contrast to the previous section) $S^{①}$ does not contain a common edge incident to v. It follows that the decision of putting H to the right or to the left of C in an embedding of $G^{①}$ has no influence on the edge ordering at v. Thus, there is no need for further constraints to ensure consistency between edge orderings and relative positions. Moreover, we will see that there is no need for additional constraints to make sure that the relative positions actually describe a face (in case v is contained in multiple cycles).

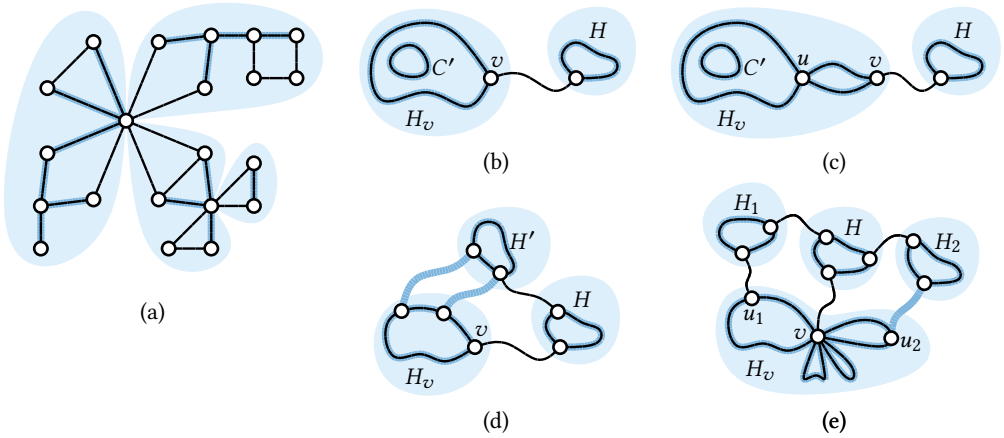


Figure 8.19: (a) The extended blocks of the graph $G^{①}$. (b) The cutvertex v and the cycle C' are contained in the same block of H_v. (c) v and C' are in different blocks. (d) The exclusive bridge containing H' has two attachments in H_v. Note that H' and H belong to the same union bridge. (e) The union bridge containing H (and H_1 and H_2) has attachments in different blocks of H_v, i.e., it is not block-local.

Sufficiency. We call the constraints defined in this section the *exclusive cutvertex constraints*. Assume that H_v is already embedded. Then we can choose to embed $S^{①}$ into an arbitrary face of H_v incident to v, which determines the relative positions of the components in $S^{①}$ with respect to cycles through v without affecting any other embedding choice. It only remains to make sure that the relative positions of H with respect to all cycles through v actually describe a face incident to v. Unfortunately, the exclusive cutvertex constraints do not guarantee this property and we are not able to give additional constraints enforcing it.

However, we can prove the following lemma by exploiting the fact that $\text{pos}_C(H)$ is in $G^{②}$ determined by an R-node, by a P-node, or by a common cutvertex. The R-node, P-node, and common cutvertex constraints of $G^{②}$ then help to prove the existence of the desired face.

We call the union of all R-node, all P-node, all common cutvertex, and all exclusive cutvertex constraints the *position constraints*.

Lemma 8.33. *Let $G^{①}$ and $G^{②}$ have common P-node degree 3 and simultaneous cutvertices of common degree at most 3. Let α be a variable assignment for the ordering and position variables satisfying the ordering and position constraints with respect to $G^{①}$ and $G^{②}$. Then $G^{①}$ admits an embedding that realizes α.*

Proof. Let the blocks of $G^{①}$ be partitioned into a maximum number of partitions such that two blocks that each have a common edge incident to a cutvertex belong to the same partition; see Figure 8.19a. Let $B_1^{①}, \ldots, B_k^{①}$ be the blocks of one such partition

and let $P^{①} = B_1^{①} \cup \cdots \cup B_k^{①}$. Note that $P^{①}$ is a maximal subgraph of $G^{①}$ such that every split component with respect to a cutvertex v includes a common edge incident to v. We call $P^{①}$ an *extended block* of $G^{①}$. By Lemmas 8.27, 8.29, 8.31, and 8.32, $P^{①}$ (and every other extended block) admits an embedding that realizes α.

Analogously, we define the extended blocks of $G^{②}$ and choose an embedding for each extended block in $G^{②}$ that realizes α.

To get an embedding of $G^{①}$ realizing α, it remains to combine the embeddings of the extended blocks at the cutvertices separating them. Let $P_v^{①}$ be the extended block of $G^{①}$ that includes a common edge incident to the cutvertex v and let $P^{①}$ be another extended block containing v. Let H_v be the connected component of the common graph containing v. Note that H_v is completely contained in $P_v^{①}$ and thus its embedding is already fixed. Note further that $P^{①}$ is part of a split component with respect to v and thus part of a single union bridge of H_v. Thus, the exclusive cutvertex constraints make sure that the relative positions with respect to common cycles through v are the same for all common connected components in $P^{①}$. Hence, the embeddings of $P_v^{①}$ and $P^{①}$ can be combined such that the resulting embedding of $P_v^{①} \cup P^{①}$ realizes α if and only if the relative positions of one common connected component H in the union bridge containing $P^{①}$ with respect to the common cycles through v describes a face of H_v that is incident to v.

We first assume that the relative positions of H with respect to cycles through v describe a face of H_v and show that this face must be incident to v. Afterwards we show that this assumption is true, making use of the fact that the common connected components are grouped differently in the extended blocks of $G^{②}$.

Claim 1. *If the relative positions of H with respect to common cycles through v describe a face of H_v, then this face is incident to v.*

Let f be a face of H_v such that the relative positions of H with respect to cycles through v (as given by α) are the same as the relative positions of f with respect to these cycles. Then f is incident to the cutvertex v for the following reason. If f is not incident to v, then there exists a common cycle C' (not containing v) in the connected component H_v of v that separates v from a connected component H. By Lemma 8.23 we can assume that C' is part of the extended cycle basis.

There are two possibilities. If v belongs in $G^{①}$ to the same block as C' (see Figure 8.19b), the relative position $\text{pos}_{C'}(H)$ is determined by the relative position of v with respect to C', as v separates H from C' in $G^{①}$ and C' does not contain v. Thus, we have the equation $\text{pos}_{C'}(v) = \text{pos}_{C'}(H)$ in this case. Otherwise, $G^{①}$ has another cutvertex u separating v and H from C'; see Figure 8.19c. Then v and H are in the same split component with respect to this cutvertex. In this case we also have the requirement that v and H are on the same side of C'. Hence, the cycle C' cannot separate v from H, which proves Claim 1. It remains to prove Claim 2.

Claim 2. *The relative positions of H with respect to all common cycles through v describe a face of H_v.*

Let $B^\cup$ be the union bridge of H_v that contains H. Recall that the exclusive cutvertex constraints require $\text{pos}_C(H') = \text{pos}_C(B^\cup)$ for every cycle C of H_v and every common connected component H' of $B^\cup$. Thus, showing that the relative positions of H' describe a face of H_v for one common connected component of $B^\cup$ shows this fact for all common connected components of $B^\cup$ (and thus in particular for H).

By Theorem 8.6, we can assume one of the following is true. The common connected component H_v is a cycle; the union bridge $B^\cup$ is not block-local; or $B^\cup$ is not exclusive one-attached.

If H_v is a cycle, then there is only a single cycle through v and both sides of this cycle form a face of H_v. Thus, there is nothing to show in this case.

Assume the union bridge $B^\cup$ is not exclusive one-attached. Then there exists without loss of generality a ②-bridge $B^②$ that belongs to the union bridge $B^\cup$ such that $B^②$ has two attachments in H_v; see Figure 8.19d. Let further H' be a common connected component contained in $B^②$. Then H' belongs in $G^②$ to a block that contains at least one block of H_v. Thus, the extended block containing H' completely contains H_v. It follows that the relative positions of H' with respect to cycles in H_v describe a face of H_v.

Finally, assume $B^\cup$ is not block-local. Then there are two ⓘ-bridges (for $i \in \{1,2\}$) $B_1^ⓘ$ and $B_2^ⓘ$ belonging to the union bridge $B^\cup$ with attachments in different blocks of H_v. If one of these bridges has an attachment vertex in a block of H_v not containing the cutvertex v, then the relative positions of this bridge with respect to any common cycle containing v is determined by an R-node, by a P-node, or by a common cutvertex. Thus, the relative positions correspond to a face of H_v in this case. It remains to consider the case that the attachment vertices of $B_1^ⓘ$ and $B_2^ⓘ$ belong to blocks of H_v incident to v; see Figure 8.19e.

Let $S_1, \dots, S_k$ be the split components of the common component H_v with respect to the cutvertex v. Assume without loss of generality that $B_1^ⓘ$ and $B_2^ⓘ$ have their attachment vertices u_1 and u_2 in S_1 and S_2, respectively. Let H_1 and H_2 be common connected components in $B_1^ⓘ$ and $B_2^ⓘ$, respectively. The relative position of H_1 with respect to a cycle through v that is not contained in S_1 is determined (in $G^ⓘ$) by the common cutvertex v. Thus, the relative positions of H_1 with respect to $S_2 \cup \dots \cup S_k$ describe a face of $S_2 \cup \dots \cup S_k$. Moreover, this face contains the whole split component S_1. Thus, if the relative positions of H_1 with respect to cycles in S_1 describe a face of S_1, then the relative positions with respect to cycles in $H_v = S_1 \cup \dots \cup S_k$ describe a face of H_v. Clearly, this is true as H_1 and H_2 have the same relative positions (they are in the same union bridge $B^\cup$) and the relative positions of H_2 with respect to cycles in S_1 describe a face of S_1 (one can use a symmetric argument to the one above). This concludes the proof. □

Computing the Constraints

Recall from Section 8.4.2 (Lemma 8.27) that we have potentially $O(n^3)$ cutvertex-ordering variables. Moreover, there are $O(n^2)$ cycles in the extended cycle basis $\mathcal{C}$ and thus $O(n^3)$ component position variables. Thus, our aim is to compute the position constraints described in the previous sections in $O(n^3)$ time.

Let $C \in \mathcal{C}$ be a cycle. For the relative positions with respect to C that are determined by R-nodes or P-nodes, we need to know for every R- and P-node μ of $G^{①}$ and of $G^{②}$, whether it induces a cycle κ in $\text{skel}(\mu)$. If so, we also need to know the cycle κ. This can clearly be done in linear time for each cycle, yielding a total running time of $O(n^3)$ (note that techniques from Chapter 7 can be used to compute this information for multiple cycles in linear time).

Similarly, in $O(n^2)$ time, we can compute for every virtual edge ε in $\text{skel}(\mu)$, which common connected components are contained in the expansion graph of ε. Assume μ is an R-node and let X be the set of ordering variables determined by μ. With the above information, one can easily compute the R-node constraints for μ in $O(|X| + |\text{skel}(\mu)|)$ time. As each relative position is determined by at most one R-node, the sets X are disjoint for different R-nodes. Thus, we get a total running time of $O(n^3)$ for computing the R-node constraints.

Computing the P-node constraints of a P-node μ can be done analogously (yielding $O(n^3)$ running time in total), except for the case where we have to handle a P-node 4-constraint. Recall that we get P-node 4-constraints if three virtual edges ε_1, ε_2, and ε_3 of $\text{skel}(\mu)$ include common paths between the poles. For every other virtual edge ε, we then get a P-node 4-constraint, which makes $O(|\text{skel}(\mu)|)$ P-node 4-constraints for μ. For the P-node 4-constraint corresponding to the virtual edge ε, we have to check whether the union graph $G^{\cup}$ has a path π from $\text{expan}(\varepsilon)$ to $\text{expan}(\varepsilon_i)$ (for $i \in \{1,2,3\}$) that is disjoint from the expansion graphs $\text{expan}(\varepsilon_j)$ for $j \in \{1,2,3\}$ with $i \neq j$. This can clearly be done in $O(n)$ time for each edge ε of $\text{skel}(\mu)$. It follows that we can compute the P-node constraints in $O(n^3)$ time.

For a cutvertex v of $G^{①}$, consider the relative positions X determined by the common cutvertex v. For every split component $S^{①}$ and every common connected component H in $S^{①}$, we have the constraint $\text{pos}_C(H) = \text{pos}_C(S^{①})$. These constraints can be easily computed in $O(n + |X|)$ time. As the sets X are disjoint for every cutvertex, this yields a total running time of $O(n^3)$. Moreover, we have to compute constraints of the type $\text{ord}(e_1, e_2, B^{①}) = \text{pos}_C(S^{①})$ connecting the relative positions to the cutvertex ordering variables. Clearly, for each variables $\text{pos}_C(S^{①})$ this constraint can be added in constant time, which yields a running time in $O(|X|)$. Hence, the common cutvertex constraints can be computed in $O(n^3)$ time.

Finally, consider the relative positions X determined by the exclusive cutvertex v and let H_v be the common connected component containing v. For every union bridge $B^{\cup}$, every common connected component H in $B^{\cup}$, and every common cycle

through v, we have to add the constraint $\text{pos}_C(B^{\cup}) = \text{pos}_C(H)$. We can first (in $O(n)$ time) partition the common connected components according to their union bridges. Then, adding these constraints for one cycle C can be done in $O(|X_C|)$ time, where X_C is the set of relative positions with respect to C in X. Thus, we get the exclusive cutvertex constraints for v in $O(|X|)$ time, which yields a total running time of $O(n^3)$.

Lemma 8.34. *The position constraints can be computed in $O(n^3)$ time.*

8.4.5 Putting Things Together

Assume $(G^{①}, G^{②})$ is a SEFE instances such that $G^{①}$ and $G^{②}$ are connected graphs, $G^{①}$ and $G^{②}$ have common P-node degree 3, and every simultaneous cutvertex has common degree 3.

We first used Theorem 8.1 to get rid of all union cutvertices. This helped to ensure consistent edge orderings in Section 8.4.2. Actually, without union cutvertices, we know for each vertex v that it is either not a cutvertex in one of the graphs $G^{①}$ or $G^{②}$, which makes representing the possible edge orderings much simpler, or that it has common degree 3, which also makes the ordering simple.

To ensure consistent relative positions of two common connected components with respect to each other, we first showed in Section 8.4.1 that it suffices to ensure consistent relative positions of each common connected component with respect to the cycles of a cycle basis in the other component. Unfortunately, setting relative positions with respect to cycles does not necessarily lead to an embedding (e.g., if a cycle C_1 lies "inside" C_2, and C_2 lies "inside" C_3, then C_3 cannot lie "inside" C_1).

This leads to difficulties, when one component H_1 can be potentially embedded into several faces of another component H_2, which is the case when H_1 is attached to H_2 via only two vertices that are a separating pair of H_2, or when H_1 and H_2 are separated by a cutvertex. For the former case, it helped to assume that split components of union separating pairs have a very special structure (Theorem 8.3). For the latter case, it helped to assume that there are no union bridges that are block-local and exclusively one-attached (Theorem 8.6).

Using Theorem 8.6 comes at the cost that we have to satisfy some common-face constraints. However, in Section 8.4.3 we showed that this can be done easily (Lemma 8.28).

The set of equations and inequalities we obtain has total $O(n^3)$ size, can be computed in $O(n^3)$ time, and can be solved in linear time in its size [APT79]. This lets us conclude with the following theorem.

Theorem 8.9. *SEFE can be solved in $O(n^3)$ time for two connected graphs with common P-node degree 3 and simultaneous cutvertices of common degree at most 3.*

8.5 Conclusion

In this chapter, we presented ways of combining techniques for ensuring consistent relative positions (Chapter 7) with known [Ang+12; BR13] and newly developed tools ensuring consistent edge orderings. This lead to an efficient algorithm solving Sefe for two connected graphs with common P-node degree 3 and simultaneous cutvertices of common degree at most 3. Together with the linear time algorithm for decomposing a given instance into equivalent instances in which each 2-component is a cycle, this gives an efficient algorithm if each connected component of the common graph is biconnected, has maximum degree 3, or is outerplanar with maximum degree 3 cutvertices.

We note that all techniques developed in Section 8.4 extend to the sunflower case, where we have multiple graphs pairwise intersecting in the same common graph. Actually, the two graphs $G^{①}$ and $G^{②}$ are only considered together if $G^{②}$ restricts the embedding choices of the common graph in $G^{①}$ in a way that makes it possible to formulate certain constraints. Thus, more graphs intersecting in the same common graph can only help. Moreover, the preprocessing algorithms from Section 8.2 also directly extend to the sunflower when adapting the definition of impossible P-nodes (Lemma 8.5) in a straightforward manner.

Besides solving this fairly general set of Sefe instances, our results, and in particular the preprocessing algorithms, give some new structural insights that may help in further research. E.g., Theorem 8.2 stating that one can assume all union-link graphs to be connected not only helps in later sections but also shows that the decision of ordering virtual edges in P-nodes of the common graph is fairly easy.

What remains poorly understood are the edge orderings at cutvertices. We were basically able to handle cutvertices if the choices boil down to binary decisions. This is for example the case if the cutvertex has only common degree 3. Although less obvious, this is also the case if the instance has common P-node degree 3. For a cutvertex in $G^{①}$, this basically means that the other graph $G^{②}$ hierarchically groups the common edges incident to the cutvertex such that there are at most three groups on each level, yielding a binary decision.

To get a better understanding of cutvertices, we believe that it can help to consider constrained planarity problems such as planarity with partitioned PQ-constrains (or variants like partitioned full-constraints); see Chapter 6.

9 Conclusion

9.1 Summary

In this thesis, I developed new approaches to long-standing open questions concerning the computational complexity of classic graph drawing problems.

Most prominently, I showed in Chapter 4 that bend minimization in the Kandinsky model is NP-hard, thereby answering a question that was open almost twenty years despite the fact that it received much attention. As an NP-hardness result is not very satisfying, I additionally gave a parameterized algorithm for the bend-minimization problem that in particular has polynomial and sub-exponential running time for series-parallel and general planar graphs, respectively. Concerning classical orthogonal drawings of graphs with maximum degree 4, I closed the complexity gap between our efficient algorithm testing whether a 4-planar graph admits an orthogonal drawing with one bend per edge [Blä+14], and the NP-hardness result by Garg and Tamassia [GT01] for the problem of finding a drawing without bends; see Chapter 2. To overcome the issue that only testing whether a graph admits a drawing with at most one bend per edge is not very useful in practice, I gave an efficient algorithm solving the corresponding optimization problem in Chapter 3. This is the first efficient algorithm capable of minimizing the number of bends in an orthogonal drawing over all planar embeddings of arbitrary 4-planar graphs. Note that this algorithm is actually of direct practical use; besides my own implementation it was also implemented in the Open Graph Drawing Framework (OGDF) [Chi+13].

In Part II, I considered the constrained planarity problems Clustered Planarity and Sefe. In Chapter 6, I provide a new perspective on the problem Clustered Planarity that simplifies and unifies many previous results, extends the set of instances that are known to be efficiently solvable, and opens up new directions for future research. Concerning simultaneous planarity, I gave the first efficient algorithm solving Sefe in a case where the relative positions of connected components with respect to each other make a difference; see Chapter 7. In Chapter 8, I combine the resulting tools for ensuring consistent relative positions with existing and newly developed techniques for ensuring consistent edge orderings. In this way, I significantly extend the set of instances that can be solved efficiently.

9.2 Outlook

The topology-shape-metrics approach in orthogonal graph drawing describes the general strategy of first fixing a planar embedding of the graph, second computing a bend-minimal orthogonal representation that respects this embedding, and third computing an area-minimal orthogonal drawing realizing this orthogonal representation. In case of non-planar graphs, the first step (topology) includes computing a planarization with few crossings. This general topology-shape-metrics strategy separates the different optimization problems from each other, making them easier to solve. In Chapter 2 and Chapter 3, I softened the strict separation between the topology and the shape step by minimizing the number of bends over all planar embeddings. This can lead to orthogonal drawings with significantly fewer bends. Concerning further research, it would be interesting to know how further softening the strict separation between the three phases can help to generate better orthogonal drawings. E.g., can we identify certain configurations in orthogonal representations that imply a large area of the resulting drawing? Can we maybe trade bends for area in the sense that adding few bends significantly shrinks the required area? Similarly, can we trade crossings in non-planar graphs for bends or even area? Transforming these questions into formal problem statements most probably results in NP-hard problems as minimizing the number of crossings and minimizing the area of a given orthogonal representation is both NP-hard. However, I showed that the moderate restriction to instances with positive flexibility allows to minimize the number of bends over all planar embeddings, which is NP-hard without this restriction. Thus, similar restrictions may also help to answer the above questions.

When considering the Kandinsky model from a practical point of view, one can generate bend-minimal drawings using an ILP-formulation. For practical instances, this usually performs very well despite the fact that it has exponential running time in the worst case. One way of explaining good practical running times is to consider parameterized algorithms whose running time is only exponential in a certain parameter, which can be assumed to be small in practical instances. I made a first step in this direction with the parameterized algorithm in Chapter 4. However, neither is the theoretical running time of this algorithm low enough to promise good practical running times, nor can it be argued that the chosen parameters are actually small in practical instances. Thus, it is an interesting open question whether there are better parameterized algorithms, maybe using completely different parameters.

When trying to solve constrained planarity problems such as Clustered Planarity or Sefe, an obstacle reoccurring in many situations is our incapability of handling the possible edge orderings at cutvertices. The problem of embedding a planar graph respecting given partitioned PQ-constraints (see Chapter 6) in a sense isolates this issue. Thus, solving this problem or one of its variants would not only solve interesting cases of Clustered Planarity but is likely to also provides insights on the

behaviour of cutvertices in other constrained planarity problems such as Sefe. As planarity with partitioned PQ-constraints is so far only solved for multi-graphs on two vertices [BKM98; HN14], results on this problem are highly interesting even if they consider only very restricted graph classes.

Another completely different way for getting a handle on the computational complexity of Sefe is the following (the same holds for Clustered Planarity). If Sefe can be solved efficiently, i.e., if Sefe is in P, then Sefe is in particular also in co-NP, i.e., for every no-instance, there exists a proof of polynomial size for being a no-instance that can be verified in polynomial time. Moreover, if one can show that Sefe is in co-NP, it is not NP-complete assuming that co-NP $\neq$ NP, which is generally believed to be true. Thus, showing that Sefe is in co-NP is weaker (and thus potentially easier) than showing that Sefe is in P. Moreover, such a result would imply that one can basically stop searching for NP-hardness proofs.

From a practical point of view, algorithms for the constrained planarity problems Clustered Planarity and Sefe are not directly applicable as they are decision problem, i.e., if there is no drawing without crossings, the algorithms return no drawings at all. Unfortunately, the corresponding optimization problems are obviously NP-hard as they include the NP-hard problem of minimizing the number of edge crossings in a graph's drawing [GJ83]. However, in my opinion, the NP-hardness of a problem should not stop researchers from studying this problem, e.g., in terms of its parameterized complexity or its approximability.

Bibliography

[AGR70] Sheldon B. Akers, James M. Geyer, and Donald L. Roberts. **IC Mask Layout with a Single Conductor Layer**. In: *Proceedings of the 7th Annual Design Automation Conference (DAC'70)*. ACM Press, 1970, 7–16.

[AL14] Patrizio Angelini and Giordano Da Lozzo. **Deepening the Relationship between SEFE and C-Planarity**. *Computing Research Repository* abs/1404.6175 (2014), 1–8.

[Ang+10] Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. **Testing Planarity of Partially Embedded Graphs**. In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*. Society for Industrial and Applied Mathematics, 2010, 202–221.

[Ang+12] Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. **Testing the Simultaneous Embeddability of Two Graphs whose Intersection is a Biconnected or a Connected Graph**. *Journal of Discrete Algorithms* 14 (2012), 150–172.

[APT79] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. **A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas**. *Information Processing Letters* 8:3 (1979), 121–123.

[BBR14] Thomas Bläsius, Guido Brückner, and Ignaz Rutter. **Complexity of Higher-Degree Orthogonal Graph Embedding in the Kandinsky Model**. In: *Proceedings of the 22th Annual European Symposium on Algorithms (ESA'14)*. Ed. by Andreas S. Schulz and Dorothea Wagner. Vol. 8737. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2014, 161–172.

[BDD00] Paola Bertolazzi, Giuseppe Di Battista, and Walter Didimo. **Computing Orthogonal Drawings with the Minimum Number of Bends**. *IEEE Transactions on Computers* 49:8 (2000), 826–840.

[BF00] Michael A. Bender and Martin Farach-Colton. **The LCA Problem Revisited**. In: *Proceedings of the 4th Latin American Symposium (LATIN'00)*. Ed. by Alfredo Viola Gaston H. Gonnet. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2000, 88–94.

[Bie98] Therese Biedl. **Drawing Planar Partitions I: LL-drawings and LH-drawings**. In: *Proceedings of the 14th Annual Symposium on Computational Geometry (SoCG'98)*. ACM Press, 1998, 287–296.

[Bin+05] Carla Binucci, Walter Didimo, Giuseppe Liotta, and Maddalena Nonato. **Orthogonal Drawings of Graphs with Vertex and Edge Labels**. *Computational Geometry: Theory and Applications* 32:2 (2005), 71–114.

[BK12] Mark de Berg and Amirali Khosravi. **Optimal Binary Space Partitions for Segments in the Plane**. *International Journal of Computational Geometry & Applications* 22:3 (2012), 187–206.

[BK98] Therese Biedl and Goos Kant. **A Better Heuristic for Orthogonal Graph Drawings**. *Computational Geometry: Theory and Applications* 9:3 (1998), 159–180.

[BKM98] Therese Biedl, Michael Kaufmann, and Petra Mutzel. **Drawing Planar Partitions II: HH-Drawings**. In: *Proceedings of the 24th Workshop on Graph-Theoretic Concepts in Computer Science (WG'98)*. Ed. by Juraj Hromkovič and Ondrej Sýkora. Vol. 1517. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1998, 124–136.

[BKR13a] Thomas Bläsius, Annette Karrer, and Ignaz Rutter. **Simultaneous Embedding: Edge Orderings, Relative Positions, Cutvertices**. In: *Proceedings of the 21nd International Symposium on Graph Drawing (GD'13)*. Ed. by Stephen Wismath and Alexander Wolff. Vol. 8242. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 220–231.

[BKR13b] Thomas Bläsius, Stephen G. Kobourov, and Ignaz Rutter. **Simultaneous Embedding of Planar Graphs**, 349–381. In: *Handbook of Graph Drawing and Visualization*. Ed. by Roberto Tamassia. Chapman and Hall/CRC, 2013.

[BL76] Kellogg S. Booth and George S. Lueker. **Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms**. *Journal of Computer and System Sciences* 13:3 (1976), 335–379.

[Blä+14] Thomas Bläsius, Marcus Krug, Ignaz Rutter, and Dorothea Wagner. **Orthogonal Graph Drawing with Flexibility Constraints**. *Algorithmica* 68:4 (2014), 859–885.

[BLR15] Thomas Bläsius, Sebastian Lehmann, and Ignaz Rutter. **Orthogonal Graph Drawing with Inflexible Edges**. In: *Proceedings of the 9th Conference on Algorithms and Complexity (CIAC'15)*. Ed. by Vangelis Th. Paschos and Peter Widmayer. Vol. 9079. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2015, 61–73.

[BMY07] Wilhelm Barth, Petra Mutzel, and Canan Yıldız. **A New Approximation Algorithm for Bend Minimization in the Kandinsky Model**. In: *Proceedings of the 14th International Symposium on Graph Drawing (GD'06)*. Ed. by Michael Kaufmann and Dorothea Wagner. Vol. 4372. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2007, 343–354.

[BNT86] Carlo Batini, Enrico Nardelli, and Roberto Tamassia. **A Layout Algorithm for Data Flow Diagrams**. *IEEE Transactions on Software Engineering* 12:4 (1986), 538–546.

[Bor+11] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. **Multiple-Source Multiple-Sink Maximum Flow in Directed Planar Graphs in Near-Linear Time**. In: *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS'11)*. IEEE Computer Society, 2011, 170–179.

[BR11] Thomas Bläsius and Ignaz Rutter. **Simultaneous PQ-Ordering with Applications to Constrained Embedding Problems**. *Computing Research Repository* abs/1112.0245 (2011), 1–46.

[BR13] Thomas Bläsius and Ignaz Rutter. **Simultaneous PQ-Ordering with Applications to Constrained Embedding Problems**. In: *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*. Society for Industrial and Applied Mathematics, 2013.

[BR14] Thomas Bläsius and Ignaz Rutter. **A New Perspective on Clustered Planarity as a Combinatorial Embedding Problem**. In: *Proceedings of the 22nd International Symposium on Graph Drawing (GD'14)*. Ed. by Christian Duncan and Antonios Symvonis. Vol. 8871. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2014, 440–451.

[BR15] Thomas Bläsius and Ignaz Rutter. **Disconnectivity and Relative Positions in Simultaneous Embeddings**. *Computational Geometry: Theory and Applications* 48:6 (2015), 459–478.

[Bra+02] Ulrik Brandes, Markus Eiglsperger, Michael Kaufmann, and Dorothea Wagner. **Sketch-Driven Orthogonal Graph Drawing**. In: *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*. Ed. by Michael T. Goodrich and Stephen G. Kobourov. Vol. 2528. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2002, 1–11.

[BRW13] Thomas Bläsius, Ignaz Rutter, and Dorothea Wagner. **Optimal Orthogonal Graph Drawing with Convex Bend Costs**. In: *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP'13)*. Ed. by Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg. Vol. 7965. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 184–195.

[BTT84] Carlo Batini, Maurizio Talamo, and Roberto Tamassia. **Computer Aided Layout of Entity Relationship Diagrams**. *Journal of Systems and Software* 4:2–3 (1984), 163–173.

[Chi+13] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. **The Open Graph Drawing Framework (OGDF)**, 543–569. In: *Handbook of Graph Drawing and Visualization*. Ed. by Roberto Tamassia. Chapman and Hall/CRC, 2013.

[Chi+14] Markus Chimani, Giuseppe Di Battista, Fabrizio Frati, and Karsten Klein. **Advances on Testing C-Planarity of Embedded Flat Clustered Graphs**. In: *Proceedings of the 22nd International Symposium on Graph Drawing (GD'14)*. Ed. by Christian Duncan and Antonios Symvonis. Vol. 8871. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2014, 416–427.

[CK12] Sabine Cornelsen and Andreas Karrenbauer. **Accelerated Bend Minimization**. *Journal of Graph Algorithms and Applications* 16:3 (2012), 635–650.

[CK13] Markus Chimani and Karsten Klein. **Shrinking the Search Space for Clustered Planarity**. In: *Proceedings of the 20th International Symposium on Graph Drawing (GD'12)*. Ed. by Walter Didimo and Maurizio Patrignani. Vol. 7704. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 90–101.

[Cor+05] Pier Francesco Cortese, Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. **Clustering Cycles into Cycles of Clusters**. *Journal of Graph Algorithms and Applications* 9:3 (2005), 391–413.

[Cor+08] Pier Francesco Cortese, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Maurizio Pizzonia. **C-Planarity of C-Connected Clustered Graphs**. *Journal of Graph Algorithms and Applications* 12:2 (2008), 225–262.

[Cor+09a] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. **Introduction to Algorithms**. 3rd. MIT Press, 2009.

[Cor+09b] Pier Francesco Cortese, Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. **On Embedding a Cycle in a Plane Graph**. *Discrete Mathematics* 309:7 (2009), 1856–1869.

[CW06] Sabine Cornelsen and Dorothea Wagner. **Completely Connected Clustered Graphs**. *Journal of Discrete Algorithms* 4:2 (2006), 313–323.

[Dah98] Elias Dahlhaus. **A Linear Time Algorithm to Recognize Clustered Planar Graphs and its Parallelization**. In: *Proceedings of the 3rd Latin American Symposium (LATIN'98)*. Ed. by Cláudio L. Lucchesi and Arnaldo V. Moura. Vol. 1380. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1998, 239–248.

[DF08] Giuseppe Di Battista and Fabrizio Frati. **Efficient C-Planarity Testing for Embedded Flat Clustered Graphs with Small Faces**. In: *Proceedings of the 15th International Symposium on Graph Drawing (GD'07)*. Ed. by Seok-Hee Hong, Takao Nishizeki, and Wu Quan. Vol. 4875. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2008, 291–302.

[DF13] Rodney G. Downey and Michael R. Fellows. **Fundamentals of Parameterized Complexity**. Springer London, 2013.

[Di +99] Giuseppe Di Battista, Walter Didimo, Maurizio Patrignani, and Maurizio Pizzonia. **Orthogonal and Quasi-Upward Drawings with Vertices of Prescribed Size**. In: *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*. Ed. by Jan Kratochvíl. Vol. 1731. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1999, 297–310.

[Die10] Reinhard Diestel. **Graph Theory**. 4th Edition. Vol. 173. Graduate Texts in Mathematics. Springer Berlin/Heidelberg, 2010.

[DLV98] Giuseppe Di Battista, Giuseppe Liotta, and Francesco Vargiu. **Spirality and Optimal Orthogonal Drawings**. *SIAM Journal on Computing* 27:6 (1998), 1764–1811.

[Dor+10] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. **Efficient Exact Algorithms on Planar Graphs: Exploiting Sphere Cut Decompositions**. *Algorithmica* 58:3 (2010), 790–810.

[DT96a] Giuseppe Di Battista and Roberto Tamassia. **On-Line Maintenance of Triconnected Components with SPQR-Trees**. *Algorithmica* 15:4 (1996), 302–318.

[DT96b] Giuseppe Di Battista and Roberto Tamassia. **On-Line Planarity Testing**. *SIAM Journal on Computing* 25:5 (1996), 956–997.

[EFK00] Markus Eiglsperger, Ulrich Fößmeier, and Michael Kaufmann. **Orthogonal Graph Drawing with Constraints**. In: *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*. Society for Industrial and Applied Mathematics, 2000, 3–11.

[Eig+04] Markus Eiglsperger, Carsten Gutwenger, Michael Kaufmann, Joachim Kupke, Michael Jünger, Sebastian Leipert, Karsten Klein, Petra Mutzel, and Martin Siebenhaller. **Automatic Layout of UML Class Diagrams in Orthogonal Style**. *Information Visualization* 3:3 (2004), 189–208.

[Eig03] Markus Eiglsperger. **Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach**. PhD thesis. Universität Tübingen, 2003.

[EIS76] Shimon Even, Alon Itai, and Adi Shamir. **On the Complexity of Timetable and Multicommodity Flow Problems**. *SIAM Journal on Computing* 5:4 (1976), 691–703.

[EK05] Cesim Erten and Stephen G. Kobourov. **Simultaneous Embedding of Planar Graphs with Few Bends**. In: *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*. Ed. by János Pach. Vol. 3383. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2005, 195–205.

[EK72] Jack Edmonds and Richard M. Karp. **Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems**. *Journal of the ACM* 19:2 (1972), 248–264.

[Fár48] István Fáry. **On Straight-Line Representation of Planar Graphs**. *Acta Scientiarum Mathematicarum* 11 (1948), 229–233.

[FCE95a] Qing-Wen Feng, Robert F. Cohen, and Peter Eades. **How to Draw a Planar Clustered Graph**. In: *Proceedings of the 1st Annual International Conference on Computing and Combinatorics (COCOON'95)*. Ed. by Ding-Zhu Du and Ming Li. Vol. 959. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1995, 21–30.

[FCE95b] Qing-Wen Feng, Robert F. Cohen, and Peter Eades. **Planarity for Clustered Graphs**. In: *Proceedings of the 3rd Annual European Symposium on Algorithms (ESA'95)*. Ed. by Paul Spirakis. Vol. 979. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1995, 213–226.

[FK95] Ulrich Fößmeier and Michael Kaufmann. **Drawing High Degree Graphs with Low Bend Numbers**. In: *Proceedings of the 3th International Symposium on Graph Drawing (GD'95)*. Ed. by Franz J. Brandenburg. Vol. 1027. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1995, 254–266.

[FK97] Ulrich Fößmeier and Michael Kaufmann. **Algorithms and Area Bounds for Nonplanar Orthogonal Drawings**. In: *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*. Ed. by Giuseppe Di Battista. Vol. 1353. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1997, 134–145.

[FKK97] Ulrich Fößmeier, Goos Kant, and Michael Kaufmann. **2-Visibility Drawings of Planar Graphs**. In: *Proceedings of the 4th International Symposium on Graph Drawing (GD'96)*. Ed. by Stephen North. Vol. 1190. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1997, 155–168.

[Fow+09] J. Joseph Fowler, Carsten Gutwenger, Michael Jünger, Petra Mutzel, and Michael Schulz. **An SPQR-Tree Approach to Decide Special Cases of Simultaneous Embedding with Fixed Edges**. In: *Proceedings of the 16th International Symposium on Graph Drawing (GD'08)*. Ed. by Ioannis G. Tollis and Maurizio Patrignani. Vol. 5417. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2009, 157–168.

[Fow+11] J. Joseph Fowler, Michael Jünger, Stephen G. Kobourov, and Michael Schulz. **Characterizations of Restricted Pairs of Planar Graphs Allowing Simultaneous Embedding with Fixed Edges**. *Computational Geometry: Theory and Applications* 44:8 (2011), 385–398.

[FT06] Fedor V. Fomin and Dimitrios M. Thilikos. **New Upper Bounds on the Decomposability of Planar Graphs**. *Journal of Graph Theory* 51:1 (2006), 53–81.

[Gas+06] Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. **Simultaneous Graph Embeddings with Fixed Edges**. In: *Proceedings of the 32nd Workshop on Graph-Theoretic Concepts in Computer Science (WG'06)*. Ed. by Fedor V. Fomin. Vol. 4271. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2006, 325–335.

[GJ79] Michael R. Garey and David S. Johnson. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. W. H. Freeman and Company, 1979.

[GJ83] Michael R. Garey and David S. Johnson. **Crossing Number is NP-Complete**. *SIAM Journal on Algebraic and Discrete Methods* 4:3 (1983), 312–316.

[GLS06] Michael T. Goodrich, George S. Lueker, and Jonathan Z. Sun. **C-Planarity of Extrovert Clustered Graphs**. In: *Proceedings of the 13th International Symposium on Graph Drawing (GD'05)*. Ed. by Patrick Healy and Nikola S. Nikolov. Vol. 3843. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2006, 211–222.

[GM01] Carsten Gutwenger and Petra Mutzel. **A Linear Time Implementation of SPQR-Trees**. In: *Proceedings of the 8th International Symposium on Graph Drawing (GD'00)*. Ed. by Joe Marks. Vol. 1984. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2001, 77–90.

[GM77] Zvi Galil and Nimrod Megiddo. **Cyclic Ordering is NP-Complete**. *Theoretical Computer Science* 5:2 (1977), 179–182.

[GT01] Ashim Garg and Roberto Tamassia. **On the Computational Complexity of Upward and Rectilinear Planarity Testing**. *SIAM Journal on Computing* 31:2 (2001), 601–625.

[GT08] Qian-Ping Gu and Hisao Tamaki. **Optimal Branch-Decomposition of Planar Graphs in $O(n^3)$ Time**. *ACM Transactions on Algorithms* 4:3 (2008), 30:1–30:13.

[GT85] Harold N. Gabow and Robert E. Tarjan. **A Linear-Time Algorithm for a Special Case of Disjoint Set Union**. *Journal of Computer and System Sciences* 30:2 (1985), 209–221.

[Gut+02] Carsten Gutwenger, Michael Jünger, Sebastian Leipert, Petra Mutzel, Merijam Percan, and René Weiskircher. **Advances in C-Planarity Testing of Clustered Graphs**. In: *Proceedings of the 10th International Symposium on Graph Drawing (GD'02)*. Vol. 2528. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2002, 220–235.

[Hen+97] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. **Faster Shortest-Path Algorithms for Planar Graphs**. *Journal of Computer and System Sciences* 55:1 (1997), 3–23.

[HJL10] Bernhard Haeupler, Krishnam Jampani, and Anna Lubiw. **Testing Simultaneous Planarity When the Common Graph Is 2-Connected**. In: *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC'10)*. Vol. 6507. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2010, 410–421.

[HJL13] Bernhard Haeupler, Krishnam Jampani, and Anna Lubiw. **Testing Simultaneous Planarity When the Common Graph Is 2-Connected**. *Journal of Graph Algorithms and Applications* 17:3 (2013), 147–171.

[HN14] Seok-Hee Hong and Hiroshi Nagamochi. **Simpler Algorithms for Testing Two-Page Book Embedding of Partitioned Graphs**. In: *Proceedings of the 20th Annual International Conference on Computing and Combinatorics (COCOON'14)*. Ed. by Zhipeng Cai, Alex Zelikovsky, and Anu Bourgeois. Vol. 8591. Lecture Notes in Computer Science. Springer International Publishing, 2014, 477–488.

[HT73] John Hopcroft and Robert E. Tarjan. **Dividing a Graph into Triconnected Components**. *SIAM Journal on Computing* 2:3 (1973), 135–158.

[HT84] Dov Harel and Robert E. Tarjan. **Fast Algorithms for Finding Nearest Common Ancestors**. *SIAM Journal on Computing* 13:2 (1984), 338–355.

[Jel+09a] Vít Jelínek, Eva Jelínková, Jan Kratochvíl, and Bernard Lidický. **Clustered Planarity: Embedded Clustered Graphs with Two-Component Clusters**. Manuscript. 2009. URL: `http://orion.math.iastate.edu/lidicky/pub/flat.pdf`.

[Jel+09b] Vít Jelínek, Eva Jelínková, Jan Kratochvíl, and Bernard Lidický. **Clustered Planarity: Embedded Clustered Graphs with Two-Component Clusters (Extended Abstract)**. In: *Proceedings of the 16th International Symposium on Graph Drawing (GD'08)*. Ed. by Ioannis G. Tollis and Maurizio Patrignani. Vol. 5417. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2009, 121–132.

[Jel+09c] Vít Jelínek, Ondřej Suchý, Marek Tesař, and Tomáš Vyskočil. **Clustered Planarity: Clusters with Few Outgoing Edges**. In: *Proceedings of the 16th International Symposium on Graph Drawing (GD'08)*. Ed. by Ioannis G. Tollis and Maurizio Patrignani. Vol. 5417. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2009, 102–113.

[Jel+09d] Eva Jelínková, Jan Kára, Jan Kratochvíl, Martin Pergel, Ondřej Suchý, and Tomáš Vyskočil. **Clustered Planarity: Small Clusters in Cycles and Eulerian Graphs**. *Journal of Graph Algorithms and Applications* 13:3 (2009), 379–422.

[JS09] Michael Jünger and Michael Schulz. **Intersection Graphs in Simultaneous Embedding with Fixed Edges**. *Journal of Graph Algorithms and Applications* 13:2 (2009), 205–218.

[KK03] Łukasz Kowalik and Maciej Kurowski. **Short Path Queries in Planar Graphs in Constant Time**. In: *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'03)*. ACM Press, 2003, 143–148.

[KM98] Gunnar W. Klau and Petra Mutzel. **Quasi-Orthogonal Drawing of Planar Graphs**. Tech. rep. Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1998.

[Kur30] Casimir Kuratowski. **Sur le Probleme des Courbes Gauches en Topologie**. *Fundamenta Mathematicae* 15 (1930), 271–283.

[Len89] Thomas Lengauer. **Hierarchical Planarity Testing Algorithms**. *Journal of the ACM* 36:3 (1989), 474–509.

[Lim14] Manuel Lima. **The Book of Trees: Visualizing Branches of Knowledge**. Princeton Architectural Press, 2014.

[Orl93] James B. Orlin. **A Faster Strongly Polynomial Minimum Cost Flow Algorithm**. *Operations Research* 41:2 (1993), 338–350.

[PCJ96] Helen C. Purchase, Robert F. Cohen, and Murray James. **Validating Graph Drawing Aesthetics**. In: *Proceedings of the 3th International Symposium on Graph Drawing (GD'95)*. Ed. by Franz J. Brandenburg. Vol. 1027. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 1996, 435–446.

[Rut11] Ignaz Rutter. **The Many Faces of Planarity: Matching, Augmentation, and Embedding Algorithms for Planar Graphs**. PhD thesis. Fakultät für Informatik, Karlsruher Institut für Technologie (KIT), 2011.

[Sch13] Marcus Schaefer. **Toward a Theory of Planarity: Hanani-Tutte and Planarity Variants**. *Journal of Graph Algorithms and Applications* 17:4 (2013), 367–440.

[ST94] Paul D. Seymour and Robin Thomas. **Call Routing and the Ratcatcher**. *Combinatorica* 14:2 (1994), 217–241.

[Sto80] James A. Storer. **The Node Cost Measure for Embedding Graphs on the Planar Grid (Extended Abstract)**. In: *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC'80)*. ACM Press, 1980, 201–210.

[Tam87] Roberto Tamassia. **On Embedding a Graph in the Grid with the Minimum Number of Bends**. *SIAM Journal on Computing* 16:3 (1987), 421–444.

[TDB88] Roberto Tamassia, Giuseppe Di Battista, and Carlo Batini. **Automatic Graph Drawing and Readability of Diagrams**. *IEEE Transactions on Systems, Man and Cybernetics* 18:1 (1988), 61–79.

[Wag36] Klaus Wagner. **Bemerkungen zum Vierfarbenproblem**. *Jahresbericht der Deutschen Mathematiker-Vereinigung* 46 (1936), 26–32.

[Wan+09] Yanju Wang, Wei-Yu Lin, Kan Liu, Rachel J. Lin, Matthias Selke, Hartmuth C. Kolb, Nangang Zhang, Xing-Zhong Zhao, Michael E. Phelps, Clifton K. F. Shen, Kym F. Faull, and Hsian-Rong Tseng. **An Integrated Microfluidic Device for Large-Scale in Situ Click Chemistry Screening**. *Lab on a Chip* 9:16 (2009), 2281–2285.

[Whi32] Hassler Whitney. **Congruent Graphs and the Connectivity of Graphs**. *American Journal of Mathematics* 54:1 (1932), 150–168.

List of Publications

Journal Articles

[1] **Testing Mutual Duality of Planar Graphs**. *International Journal of Computational Geometry & Applications* 24:4 (2015), 325–346. Joint work with Patrizio Angelini and Ignaz Rutter.

[2] **Orthogonal Graph Drawing with Flexibility Constraints**. *Algorithmica* 68:4 (2014), 859–885. Joint work with Marcus Krug, Ignaz Rutter, and Dorothea Wagner.

[3] **Disconnectivity and Relative Positions in Simultaneous Embeddings**. *Computational Geometry: Theory and Applications* 48:6 (2015), 459–478. Joint work with Ignaz Rutter.

[4] **Simultaneous PQ-Ordering with Applications to Constrained Embedding Problems**. *ACM Transactions on Algorithms* (2015). Accepted for Publication. Joint work with Ignaz Rutter.

Articles in Refereed Conference Proceedings

[5] **Pixel and Voxel Representations of Graphs**. In: *Proceedings of the 23rd International Symposium on Graph Drawing (GD'15)*. Lecture Notes in Computer Science. Accepted for Publication. Springer Berlin/Heidelberg, 2015. Joint work with M. Jawaherul Alam, Ignaz Rutter, Torsten Ueckerdt, and Alexander Wolff.

[6] **Testing Mutual Duality of Planar Graphs**. In: *Proceedings of the 24th International Symposium on Algorithms and Computation (ISAAC'13)*. Ed. by Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam. Vol. 8283. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 350–360. Joint work with Patrizio Angelini and Ignaz Rutter.

[7] **Using ILP/SAT to Determine Pathwidth, Visibility Representations, and other Grid-Based Graph Drawings**. In: *Proceedings of the 21st International Symposium on Graph Drawing (GD'13)*. Ed. by Stephen Wismath and Alexander Wolff. Vol. 8242. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 460–471. Joint work with Therese Biedl, Benjamin Niedermann, Martin Nöllenburg, Roman Prutkin, and Ignaz Rutter.

[8] **Complexity of Higher-Degree Orthogonal Graph Embedding in the Kandinsky Model**. In: *Proceedings of the 22th Annual European Symposium on Algorithms (ESA'14)*. Ed. by Andreas S. Schulz and Dorothea Wagner. Vol. 8737. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2014, 161–172. Joint work with Guido Brückner and Ignaz Rutter.

[9] **Simultaneous Embedding: Edge Orderings, Relative Positions, Cutvertices**. In: *Proceedings of the 21nd International Symposium on Graph Drawing (GD'13)*. Ed. by Stephen Wismath and Alexander Wolff. Vol. 8242. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 220–231. Joint work with Annette Karrer and Ignaz Rutter.

[10] **Orthogonal Graph Drawing with Flexibility Constraints**. In: *Proceedings of the 18th International Symposium on Graph Drawing (GD'10)*. Ed. by Ulrik Brandes and Sabine Cornelsen. Vol. 6502. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2011, 92–104. Joint work with Marcus Krug, Ignaz Rutter, and Dorothea Wagner.

[11] **Orthogonal Graph Drawing with Inflexible Edges**. In: *Proceedings of the 9th Conference on Algorithms and Complexity (CIAC'15)*. Ed. by Vangelis Th. Paschos and Peter Widmayer. Vol. 9079. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2015, 61–73. Joint work with Sebastian Lehmann and Ignaz Rutter.

[12] **Disconnectivity and Relative Positions in Simultaneous Embeddings**. In: *Proceedings of the 20th International Symposium on Graph Drawing (GD'12)*. Ed. by Walter Didimo and Maurizio Patrignani. Vol. 7704. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2013, 31–42. Joint work with Ignaz Rutter.

[13] **Simultaneous PQ-Ordering with Applications to Constrained Embedding Problems**. In: *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*. Society for Industrial and Applied Mathematics, 2013. Joint work with Ignaz Rutter.

[14] **A New Perspective on Clustered Planarity as a Combinatorial Embedding Problem**. In: *Proceedings of the 22nd International Symposium on Graph Drawing (GD'14)*. Ed. by Christian Duncan and Antonios Symvonis. Vol. 8871. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2014, 440–451. Joint work with Ignaz Rutter.

[15] **Optimal Orthogonal Graph Drawing with Convex Bend Costs**. In: *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP'13)*. Ed. by Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg. Vol. 7965. Lecture Notes in Computer Sci-

ence. Springer Berlin/Heidelberg, 2013, 184–195. Joint work with Ignaz Rutter and Dorothea Wagner.

Book Chapters

[16] **Simultaneous Embedding of Planar Graphs**, 349–381. In: *Handbook of Graph Drawing and Visualization*. Ed. by Roberto Tamassia. Chapman and Hall/CRC, 2013. Joint work with Stephen G. Kobourov and Ignaz Rutter.

Theses

[17] **Orthogonal Graph Drawing with Flexibility Constraints**. Study Thesis. Faculty of Informatics, Karlsruhe Institute of Technology (KIT), Aug. 2010.

[18] **Simultaneous PQ-Ordering with Applications to Constrained Embedding Problems**. Diploma Thesis. Faculty of Informatics, Karlsruhe Institute of Technology (KIT), Sept. 2011.

Poster

[19] **PIGRA – A Tool for Pixelated Graph Representations**. In: *Proceedings of the 22nd International Symposium on Graph Drawing (GD'14)*. Ed. by Christian A. Duncan and Antonios Symvonis. Vol. 8871. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2014, 513–514. Joint work with Fabian Klute, Benjamin Niedermann, and Martin Nöllenburg.