Partitioning Subclasses of Chordal Graphs with Few Deletions

Satyabrata Jana[1[0000−0002−7046−0091]], Souvik Saha[1[0000−0001−8322−0639]], Abhishek Sahu[3], Saket Saurabh[1,2], and Shaily Verma[1]

[1] The Institute of Mathematical Sciences, HBNI, Chennai, India
[2] University of Bergen, Norway
[3] National Institute of Science Education and Research, HBNI
{souviks, saket, shailyverma}@imsc.res.in
{satyamtma, asahuiitkgp}@gmail.com

Abstract. In the (Vertex) k-Way Cut problem, input is an undirected graph G, an integer s, and the goal is to find a subset S of edges (vertices) of size at most s, such that $G - S$ has at least k connected components. Downey et al. [Electr. Notes Theor. Comput. Sci. 2003] showed that k-Way Cut is W[1]-hard parameterized by k. However, Kawarabayashi and Thorup [FOCS 2011] showed that the problem is fixed-parameter tractable (FPT) in general graphs with respect to the parameter s and provided a $\mathcal{O}(s^{s^{\mathcal{O}(s)}} n^2)$ time algorithm, where n denotes the number of vertices in G. The best-known algorithm for this problem runs in time $s^{\mathcal{O}(s)} n^{\mathcal{O}(1)}$ given by Lokshtanov et al. [ACM Tran. of Algo. 2021]. On the other hand, Vertex k-Way Cut is W[1]-hard with respect to either of the parameters, k or s or $k + s$. These algorithmic results motivate us to look at the problems on special classes of graphs.

In this paper, we consider the (Vertex) k-Way Cut problem on subclasses of chordal graphs and obtain the following results.

– We first give a sub-exponential FPT algorithm for k-Way Cut running in time $2^{\mathcal{O}(\sqrt{s}\log s)} n^{\mathcal{O}(1)}$ on chordal graphs.
– It is "known" that Vertex k-Way Cut is W[1]-hard on chordal graphs, in fact on split graphs, parameterized by $k + s$. We complement this hardness result by designing polynomial-time algorithms for Vertex k-Way Cut on interval graphs, circular-arc graphs and permutation graphs.

Keywords: chordal graphs · FPT · interval graphs · circular-arc graphs· permutation graphs.

1 Introduction

Graph partitioning problems have been extensively studied because of their applications in VLSI design, parallel supercomputing, image processing, and clustering [1]. In this paper, we consider one of the classical graph partitioning problems, namely, the (Vertex) k-Way Cut problem. In this problem the objective is to partition the graph into k components by deleting as few (vertices) edges as possible. Formally, the problems we study are defined as follows.

k-Way Cut

Input: A graph $G = (V, E)$ and two integers s and k.
Parameter: s
Question: Does there exist a set $S \subseteq E$ of size at most s, such that $G - S$ has at least k connected components?

Vertex k-Way Cut

Input: A graph $G = (V, E)$ and two integers s and k.
Parameter: s
Question: Does there exist a set $S \subseteq V$ of size at most s, such that $G - S$ has at least k connected components?

These problems are decision versions of natural generalization of the Global Min Cut problem, which seeks to delete a set of edges of minimum cardinality such that the graph gets partitioned into two parts ($k = 2$). In other words, the graph becomes disconnected. We first give a brief account of the history of known results on the problem to set the context of our study.

Algorithmic History of the Problem. There is a rich algorithmic study of (Vertex) k-Way Cut problem. In 1996, Goldschmidt and Hochbaum [6] showed that the k-Way Cut problem is NP-hard for arbitrary k, but polynomial-time solvable when k is fixed and gave a $\mathcal{O}(n^{(1/2-o(1))k^2})$ time algorithm, where n is the number of vertices in the graph. Later, Karger and Stein [10] gave an edge contraction based randomized algorithm with running time $\tilde{\mathcal{O}}(n^{(2k-1)})$. The notation $\tilde{\mathcal{O}}$ hides the poly-logarithimic factor in the running time. Recently, Li [13] obtained an improved randomized algorithm with running time $\tilde{\mathcal{O}}(n^{(1.981+o(1))k})$. To date, the best known deterministic exact algorithm is given by Chekuri et al. [2] which runs in $\mathcal{O}(mn^{(2k-3)})$ time.

In terms of approximation algorithms, several approximation algorithms are known for the k-Way Cut problem with approximation factor $(2 - o(1))$, that run in time polynomial in n and k [17] Recently, Manurangsi [15] proved that the approximation factor cannot be improved to $(2 - \epsilon)$ for every $\epsilon > 0$, assuming small set expansion hypothesis. Lately, this problem has received significant attention from the perspective of parameterized approximation as well. Gupta et al. [8] gave the first FPT approximation algorithm for the problem with approximation factor 1.9997 which runs in time $2^{\mathcal{O}(k^6)}n^{\mathcal{O}(1)}$. The same set of authors [9] also gave an $(1 + \epsilon)$-approximation algorithm with running time $(k/\epsilon)^{\mathcal{O}(k)}n^{k+\mathcal{O}(1)}$, and an approximation algorithm with a factor 1.81 running in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$. Later, Kawarabayashi and Lin [11] gave a $(5/3 + \epsilon)$-approximation algorithm for the problem with running time $2^{\mathcal{O}(k^2 \log k)}n^{\mathcal{O}(1)}$. Recently, Lokshtanov et al. [14] designed $(1 + \epsilon)$-approximation algorithm for every $\epsilon > 0$, running in time $(k/\epsilon)^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ improving upon the previous result.

Problems	Parameter(s)		
	k	s	$k+s$
Vertex k-Way Cut	W[1]-hard [5]	W[1]-hard [16]	W[1]-hard [16]
k-Way Cut	W[1]-hard [5]	FPT [12]	FPT [4]

Table 1. Complexity of the problems for different parameterizations

From the parameterized perspective, Downey et al. [5] proved that the k-Way Cut and Vertex k-Way Cut problems are W[1]-hard when parameterized by k. On the other hand, when parameterized by the cut size s, it is known that finding a Vertex k-Way Cut of size s is also W[1]-hard [16]; however finding a k-Way Cut of size s is FPT [12]. Kawarabayashi and Thorup [12] gave a $\mathcal{O}(s^{s^{\mathcal{O}(s)}} \cdot n^2)$ time FPT algorithm for the k-Way Cut problem. Recently, Lokshtanov et al. [4] designed a faster algorithm with running time $s^{\mathcal{O}(s)}n^{\mathcal{O}(1)}$. These tractable and intractable results (see Table 1) are a starting point of our work. That is, we address the following question: *What is the complexity of* (Vertex) k-Way Cut *problem on well-known graph classes?*

Our Results. In this paper we obtain a a sub-exponential-FPT algorithm for k-Way Cut running in time $2^{\mathcal{O}(\sqrt{s}\log s)}n^{\mathcal{O}(1)}$ on chordal graphs (Section 3) and polynomial-time algorithms for Vertex k-Way Cut on interval graphs, circular-arc graphs, and permutation graphs (Section 4).

2 Preliminaries

All graphs considered in this paper are finite, simple, and undirected. We use the standard notation and terminology that can be found in the book of graph theory [18]. We use $[n]$ to denote the set of first n positive integers $\{1, 2, 3, \ldots, n\}$. For a graph G, we denote the set of vertices of the graph by $V(G)$ and the set of edges of the graph by $E(G)$. We denote $|V(G)|$ and $|E(G)|$ by n and m respectively, where the graph is clear from context. We abbreviate an edge (u, v) as uv sometimes. For a set $S \subseteq V(G)$, the subgraph of G induced by S is denoted by $G[S]$ and it is defined as the subgraph of G with vertex set S and edge set $\{(u, v) \in E(G) : u, v \in S\}$ and the subgraph obtained after deleting S (and the edges incident to the vertices in S) is denoted by $G - S$. For $v \in V(G)$, we will use $G - v$ to denote $G - \{v\}$ for ease of notation. All vertices adjacent to a vertex v are called neighbours of v and the set of all such vertices is called the open neighbourhood of v, denoted by $N_G(v)$. For a set of vertices $S \subseteq V(G)$, we define $N_G(S) = (\cup_{v\in S} N(v) \setminus S)$. We define the closed neighbourhood of a vertex v in the graph G to be $N_G[v] := N_G(v) \cup \{v\}$ and closed neighbourhood of a set of vertices $S \subseteq V(G)$ to be $N_G[S] := N_G(S) \cup S$. We drop the subscript G when the graph is clear from the context. For $C \subseteq V(G)$, if $G[C]$ is connected and $N(C) = \emptyset$, then we say that $G[C]$ is a connected component of G. For both the problems k-Way Cut and Vertex k-Way Cut, in the given instance, we assume that $k > 1$, otherwise the input itself is an optimal solution with zero cut size. A partition of G in to k components is a partition of $V(G)$ into k sets $V_1, \ldots, V_k$ such that each $G[V_i]$ is a connected. We say a partition is *non-trivial* when $k > 1$.

Definition 1. *A tree-decomposition of a connected graph G is a pair (T, β), where T is a tree and and $\beta \colon V(T) \to V(G)$ such that*

- $\bigcup_{x \in V(T)} \beta(x) = V(G)$*, we call $\beta(x)$ as the bag of x,*
- *for every edge $(u, v) \in E(G)$, there exists $x \in V(T)$ such that $\{u, v\} \subseteq \beta(x)$, and*
- *for every vertex $v \in V(G)$, the subgraph of T induced by the set $\beta^{-1}(v) := \{x \colon v \in \beta(x)\}$ is connected.*

Chordal Graphs: A graph G is a *chordal graph* if every cycle in G of length at least 4 has a *chord* i.e., an edge joining two non-consecutive vertices of the cycle. A *clique-tree* of G is a tree-decomposition of G where every bag is a maximal clique. We further insist that every bag of the clique-tree is distinct. There are several ways to obtain a clique-tree decomposition of G; one way is by using perfect elimination ordering (PEO) of G [3]. The following lemma shows that the class of chordal graphs is exactly the class of graphs that have a clique-tree.

Lemma 1 ([7]). *A connected graph G is a chordal graph if and only if G has a clique-tree.*

Let $\mathscr{F}$ be a non-empty family of sets. A graph G is called an *intersection graph* for $\mathscr{F}$ if there is a one-to-one correspondence between $\mathscr{F}$ and G where two sets in $\mathscr{F}$ have nonempty intersection if and only if their corresponding vertices in G are adjacent. We call $\mathscr{F}$ an *intersection model* of G and we use $G(\mathscr{F})$ to denote the intersection graph for $\mathscr{F}$. If $\mathscr{F}$ is a family of intervals on a real line, then $G(\mathscr{F})$ is called an *interval graph* for $\mathscr{F}$. A *proper interval graph* is an interval graph that has an intersection model in which no interval properly contains another. If $\mathscr{F}$ is a family of arcs on a circle in the plane, then $G(\mathscr{F})$ is called an *circular-arc graph* for $\mathscr{F}$. If $\mathscr{F}$ is a family of line segments in the plane whose endpoints lie on two parallel lines, then the intersection graph of $\mathscr{F}$ is called the *permutation graph* for $\mathscr{F}$.

3 Sub-exponential FPT Algorithm on Chordal Graphs

Chordal graphs belong to the class of perfect graphs that contains several other graph classes such as split graphs, interval graphs, threshold graphs, and block graphs. A graph G is a *chordal graph* if every cycle in G of length at least 4 has a *chord* i.e., an edge joining two non-consecutive vertices of the cycle. Chordal graphs are also characterized as the intersection graph of sub-trees of a tree. Every chordal graph has a tree-decomposition where every bag induces a clique. In this section, we obtain a sub-exponential FPT algorithm for the k-Way Cut problem in chordal graphs parameterized by s, the number of cut edges. We first give a characterization of the k-Way Cut on a clique in Lemma 3. Later, we use this characterization to design our algorithm.

Lemma 2. *Let $\mathbb{K}$ be a clique and s be an integer. Then we can not partition the clique into more than one component by deleting s edges if one of the following conditions holds.*

(i) $|\mathbb{K}| > (s+1)$,

(ii) $|\mathbb{K}| > (2\sqrt{s}+1)$, *and size of every component in the partition is at most* $\sqrt{s}$.

Proof. (i) If $|\mathbb{K}| > (s+1)$, the size of min-cut of $\mathbb{K}$ is at least $s+1$ and hence we cannot partition $\mathbb{K}$ by deleting s edges. (ii) In the second condition, the size of every component in the partition is at most $\sqrt{s}$ and hence every vertex v in any component must be disconnected from at least $2\sqrt{s}+2-\sqrt{s} = \sqrt{s}+2$ vertices that are in other components. Thus the total number of edges that needs to be deleted is at least $(2\sqrt{s}+2)(\sqrt{s}+2)/2 > s$. Hence the clique can not be partitioned by deleting s edges. ◻

Lemma 3. *Let $\mathbb{K}$ be a clique and s be an integer such that $(2\sqrt{s}+1) < |\mathbb{K}| < (s+2)$, then any non-trivial partition of $\mathbb{K}$ obtained by deleting at most s edges, has a component of size at least $(|\mathbb{K}| - \sqrt{s})$.*

Proof. Let $\mathbb{K}$ be a clique such that $(2\sqrt{s}+1) < |\mathbb{K}| < (s+2)$ and we have to partition the clique into k components by deleting at most s edges. Let γ be the size of the largest component in the partition.

$$\begin{aligned}
&|E(\mathbb{K})| = |E(\text{Largest component})| + |\text{E(other components)}| + |\text{cut edges}| \\
&\implies \binom{|\mathbb{K}|}{2} \le \binom{\gamma}{2} + \binom{|\mathbb{K}|-\gamma}{2} + |\text{cut edges}| \\
&\implies \binom{|\mathbb{K}|}{2} \le \binom{\gamma}{2} + \binom{|\mathbb{K}|-\gamma}{2} + s \\
&\implies |\mathbb{K}|(|\mathbb{K}|-1) \le \gamma(\gamma-1) + (|\mathbb{K}|-\gamma)(|\mathbb{K}|-\gamma-1) + 2s \\
&\implies 0 \le \gamma^2 - \gamma|\mathbb{K}| + s
\end{aligned}$$

Therefore, either $\gamma \le \frac{|\mathbb{K}|-\sqrt{|\mathbb{K}|^2-4s}}{2}$, or $\gamma \ge \frac{|\mathbb{K}|+\sqrt{|\mathbb{K}|^2-4s}}{2}$ holds. If the first inequality holds, then it implies $\gamma \le \frac{|\mathbb{K}|-\sqrt{|\mathbb{K}|^2}+\sqrt{4s}}{2}$ (by using the inequality $\sqrt{a}-\sqrt{b} \le \sqrt{a-b}$ *for* $0 < b \le a$). It follows that $\gamma \le \sqrt{s}$. However, Lemma 2 implies that if $\gamma \le \sqrt{s}$ and $|\mathbb{K}| > 2\sqrt{s}+1$, then there is no non-trivial partition of $\mathbb{K}$. Thus in this case, $\mathbb{K}$ has no non-trivial partition. If the second inequality holds, then $\gamma \ge \frac{|\mathbb{K}|+\sqrt{|\mathbb{K}|^2-4s}}{2}$, which implies that $\gamma \ge (|\mathbb{K}| - \sqrt{s})$. Hence any non-trivial partition of $\mathbb{K}$, obtained by deleting at most s edges, has a component of size at least $(|\mathbb{K}| - \sqrt{s})$. ◻

Lemma 4. *There are $2^{\mathcal{O}(\sqrt{s}\log s)}$ many possible choices for any non-trivial partition of a clique $\mathbb{K}$ obtained by deleting at most s edges.*

Proof. We have the following three cases depending on the size of $\mathbb{K}$.

Case 1. $|\mathbb{K}| \ge (s+2)$.

In this case, no non-trivial partition exists by Lemma 2.

Case 2. $|\mathbb{K}| \leq (2\sqrt{s}+1)$.

In this case, there are $k^{2\sqrt{s}+1}$ ways of partitioning the clique into k components. Since $k \leq (s+1)$, $k^{2\sqrt{s}+1} \leq 2^{\mathcal{O}(\sqrt{s}\log s)}$.

Case 3. $(2\sqrt{s}+1) < |\mathbb{K}| < (s+2)$.

From Lemma 3, in a partition of $\mathbb{K}$ into k components, there exists a component with at least $(|\mathbb{K}| - \sqrt{s})$ many vertices. So, we guess $(|\mathbb{K}| - \sqrt{s})$ many vertices in a component. Now, the rest $\sqrt{s}$ vertices are partitioned into k components. The total number of choices for such a partition of $\mathbb{K}$ is bounded by $\binom{|\mathbb{K}|}{|\mathbb{K}|-\sqrt{s}} \cdot k^{\sqrt{s}} \cdot k$. Since both k and $|\mathbb{K}|$ are bounded by $(s+1)$, we have $|\mathbb{K}|^{\sqrt{s}} \cdot k^{\sqrt{s}} \cdot k \leq 2^{\mathcal{O}(\sqrt{s}\log s)}$.

□

Now we prove the following theorem.

Theorem 1. *k-Way Cut problem on a chordal graph with n vertices can be solved in time $2^{\mathcal{O}(\sqrt{s}\log s)} n^{\mathcal{O}(1)}$.*

To prove Theorem 1, we design a dynamic-programming algorithm for the k-Way Cut problem on chordal graphs, which exploits its clique-tree decomposition. Let G be a chordal graph and $\mathcal{T} = (T, \{K_t\}_{t\in V(t)})$ be its clique-tree decomposition.

Let T be a clique-tree of G rooted at some node r. For a node t of T, K_t is the set of vertices contained in t and let V_t be the set of all vertices of the sub-tree of T rooted at t. The parent node of t is denoted by $\mathsf{parent}(t)$. We follow a bottom-up dynamic-programming approach on T to design our algorithm.

For a set of vertices U, we use $\mathtt{P}(U)$ to denote a partition $\{A_1, A_2, \ldots, A_k\}$ of U where each A_i is a set in the partition. Given the partitions of two sets $U_1, U_2 \subseteq V(G)$, say $\mathtt{P}(U_1) = \{A_1, A_2, \ldots, A_k\}$ and $\mathtt{P}(U_2) = \{B_1, B_2, \ldots, B_k\}$, we call these partitions *mutually compatible*, if for each vertex u in $U_1 \cap U_2$, $u \in A_i$ if and only if $u \in B_i$ for some $i \in [k]$. We denote the mutually compatible operation by $\perp$. For any node t, a partition $\mathtt{P}(K_t)$ and an integer w where $0 \leq w \leq (k-1)$, a feasible solution for $(t, \mathtt{P}(K_t), w)$ is a k-way cut in $G[V_t]$ with the following properties: ($\mathtt{P}(V_t)$ is the partition induced on V_t by the above k-way cut).

- $\mathtt{P}(K_t) \perp \mathtt{P}(V_t)$,
- Exactly w components in $\mathtt{P}(V_t)$ contain no vertex from K_t, that is, these w components are completely contained inside $G[V_t \setminus K_t]$.

Next, we define the dynamic-programming table whose entry is denoted by $M[t; \mathtt{P}(K_t), w]$ for a node t and integer w, $0 \leq w \leq k$. The entry $M[t; \mathtt{P}(K_t), w]$ stores the size of the smallest such feasible solution. From Lemma 4, the number of sub-problems (or number of entries that we have to compute) for each node in the tree is bounded by $2^{\mathcal{O}(\sqrt{s}\log s)}$ as each node is a clique. Below we give a recurrence relation to compute $M[t; \mathtt{P}(K_t), w]$ for each tuple $(t, \mathtt{P}(K_t), w)$. The case where t is a leaf, corresponds to the base case of the recurrence, whereas the values of $M[t;.,.]$ for a non-leaf node t depends on the value of $M[t',.]$ for each child t' of node t (which have already been computed). By applying the formula in a bottom-up manner on T, we compute $M(r;\mathtt{P}(K_r),k-1)$ for the root node r. Note that the value of $M(r;\mathtt{P}(K_r),k-1)$ is exactly the size of an optimal solution for our problem, because in any optimal solution there are exactly $k-1$ components that are completely contained in $G-K_r$. Here without loss of generality, we can assume that K_r contains exactly one vertex of G. For a partition $\mathtt{P}(U)$ of U, we define $\mathtt{CUT}(\mathtt{P}(U))$ as the set of edges whose endpoints belong to different sets in the partition. Now, we describe the recursive formulas to compute the value of $M[t;.,.]$, for each node t.

Leaf node. Let t be a leaf node. Then for each partition $\mathtt{P}(K_t)$, we define

$$M[t;\mathtt{P}(K_t),w] = \begin{cases} |\mathtt{CUT}(\mathtt{P}(K_t))| & \text{if } w=0, \\ +\infty & \text{otherwise.} \end{cases}$$

Non-leaf node. Let t be a non-leaf node. Assume that the node t has ℓ children $t_1,\ldots,t_\ell$. For a pair of distinct vertices u,v in K_t, let $\mathtt{Child\_Pair}(t;u,v)$ denote the number of children of t containing both the vertices u and v. For a partition $\mathtt{P}(K_t)$, let $\mathtt{Child}(\mathtt{P}(K_t))$ denote the sum of the number of occurrences (with repetitions) of the edges from $\mathtt{CUT}(\mathtt{P}(K_t))$ in all the children nodes of t, that is, $\mathtt{Child}(\mathtt{P}(K_t)) = \sum_{(u,v)\in\mathtt{CUT}(\mathtt{P}(K_t))} \mathtt{Child\_Pair}(t;u,v)$. Let $\psi(\mathtt{P}(K_t))$ denote the number of sets in $\mathtt{P}(K_t)$ that have no common vertex with the parent node of t. Therefore, the recurrence relation for computing $M(t;.,.)$ for t is as follows:

$$\begin{aligned} M[t;\mathtt{P}(K_t),w] = {} & |\mathtt{CUT}(\mathtt{P}(K_t))| - \mathtt{Child}(\mathtt{P}(K_t)) + \\ & \min_{\substack{\forall(\mathtt{P}(K_{t_i}),w_i): \\ \mathtt{P}(K_{t_i})\perp\mathtt{P}(K_t) \\ w=\sum_i(w_i+\psi(\mathtt{P}(K_{t_i})))}} \sum_{i=1}^{\ell} M[t_i;\mathtt{P}(K_{t_i}),w_i]. \end{aligned}$$

Next, we prove the correctness of the above recurrence relation.

Correctness. Let R denote the value of the right side expression above. To prove the recurrence relation, first we show $M[t;\mathtt{P}(K_t),w] \le R$ and then $M[t;\mathtt{P}(K_t),w] \ge R$. Let t be a node in T having ℓ children $t_1,t_2,\ldots,t_\ell$. Any set of ℓ compatible partitions, one for each child of t together with $\mathtt{P}(K_t)$ leads to a feasible solution for $(t,\mathtt{P}(K_t),w)$ if $w=\sum_i(w_i+\psi(\mathtt{P}(K_{t_i})))$. Now for each child node t_i of t and for any pair of vertices u,v in K_t, if the vertices u and v are in different sets in each of the partitions $\mathtt{P}(K_t)$ and $\mathtt{P}(K_{t_i})$, then the (to be deleted) edge (u,v) is counted twice, once in $\mathtt{CUT}(\mathtt{P}(K_t))$ and once in $M[t_i;\mathtt{P}(K_{t_i}),w_i]$. Now if the edge (u,v) is present in c many children of t, then in the entry $\sum\limits_{i=1}^{\ell} M[t_i;\mathtt{P}(K_{t_i}),w_i]$ this edge gets counted c times. To avoid over-counting of the edge (u,v) in $M[t_i;.,.]$, we must consider the edge (u,v) exactly once and for this purpose we use $\mathtt{Child}(\mathtt{P}(K_t))$ in the recurrence relation. Considering this over counting, the set of edges corresponding to $M[t_1;\mathtt{P}(K_{t_1}),w_1], M[t_2;\mathtt{P}(K_{t_2}),w_2],\ldots,M[t_\ell;\mathtt{P}(K_{t_\ell}),w_\ell]$ with size $\sum_{i=1}^{\ell} M[t_i;\mathtt{P}(K_{t_i}),w_i] - \mathtt{Child}(\mathtt{P}(K_t))$, together with the edges corresponding to $\mathtt{CUT}(\mathtt{P}(K_t))$ gives us a feasible solution for $(t,\mathtt{P}(K_t),w)$. Hence, $M[t;\mathtt{P}(K_t),w] \leq |\mathtt{CUT}(\mathtt{P}(K_t))| - \mathtt{Child}(\mathtt{P}(K_t)) + \sum_{i=1}^{\ell} M[t_i;\mathtt{P}(K_{t_i}),w_i]$, where $\mathtt{P}(K_t) \perp \mathtt{P}(K_{t_i})$ for each $i \in [\ell]$ and $w = \sum_i (w_i + \psi(\mathtt{P}(K_{t_i}))$.

Next, we show that $M[t;\mathtt{P}(K_t),w] \geq R$. Let Y be a set of cut edges corresponding to the entry $M[t;\mathtt{P}(K_t),w]$. Let $Y' \subseteq Y$ be the set of edges that are not present in K_t. So $Y \setminus Y'$ determines the partition in K_t. Let $Y' = Y_1 \cup \ldots \cup Y_\ell$, where each Y_i is the set of edges for $G[V(t_i)]$. Let $X_1 \cup \ldots \cup X_\ell \subseteq (Y \setminus Y')$, where $X_i = (Y \setminus Y') \cap E(K(t_i))$. Now it is easy to see that $Y_i \cup X_i$ is a feasible solution for $(t_i,\mathtt{P}(K_{t_i}),w_i)$, where $\mathtt{P}(K_t) \perp \mathtt{P}(K_{t_i})$ for each $i \in [\ell]$ and $w = \sum_i (w_i + \psi(\mathtt{P}(K_{t_i}))$. Since $Y \setminus Y'$ determines the partition only in K_t, $|Y \setminus Y'| = |\mathtt{CUT}(\mathtt{P}(K_t))|$. Thus, we get $M[t;\mathtt{P}(K_t),w] - |\mathtt{CUT}(\mathtt{P}(K_t))| + \mathtt{Child}(\mathtt{P}(K_t)) \geq \sum_{i=1}^{\ell} M[t_i;\mathtt{P}(K_{t_i}),w_i]$. Hence the correctness of the recurrence relation follows.

Time complexity. There are $\mathcal{O}(n)$ many nodes in the clique tree of the given graph G. The number of entries $M[.;.,.]$ for any node can be upper bounded by $k2^{\mathcal{O}(\sqrt{s}\log s)}$ (from Lemma 4). To compute one such entry, we look at the entries with the compatible partitions in the children nodes. Now, we describe how we compute $M[t;\mathtt{P}(K_t),w]$ in a node for a fixed partition $\mathtt{P}(K_t)$ and a fixed integer $w \leq k$. We apply an incremental procedure to find this. Consider an ordering $t_1 \prec t_2 \prec \ldots \prec t_\ell$ of child nodes of t. In the dynamic-programming, we store the entries $M[t_i;\mathtt{P}(K_{t_i}),w_i]$ for each $\mathtt{P}(K_{t_i}) \perp \mathtt{P}(K_t)$ and $w_i \leq k$. For each t_i, we compute the entries $D_i(z)$ for $0 \leq z \leq k$, where $D_i(z) = \min_{z} \{M[t_i;\mathtt{P}(K_{t_i}),w^*] : \mathtt{P}(K_{t_i}) \perp \mathtt{P}(K_t),\ z = w^* + \psi(\mathtt{P}(K_{t_i}), w^* \leq k\}$. Next we create a set of entries for D, defined by
$D(1,2,\ldots,i;z) = \min_{z=z_1+z_2} \{D(1,2,\ldots,i-1;z_1) + D_i(z_2)\}$, for $i \in [\ell]$. $D(1;z) = D_1(z), \forall z$ (the base case). It takes $\mathcal{O}(\ell k^3)$ time to compute all the entries of the table D. Now using the entries of the table D, we compute $M[t;\mathtt{P}(K_t),w]$, i.e.
$M[t;\mathtt{P}(K_t),z] = |\mathtt{CUT}(\mathtt{P}(K_t))| - \mathtt{Child}(\mathtt{P}(K_t)) + D(1,2,\ldots,\ell;z)$.
Since there are $2^{\mathcal{O}(\sqrt{s}\log s)}$ many partitions of each node t, computing all DP table entries at each node takes $2^{\mathcal{O}(\sqrt{s}\log s)}\mathcal{O}(\ell k^3)$ time. Because $\ell, k \leq n$, and there are $\mathcal{O}(n)$ many nodes in the clique tree, the total running time is upper-bounded by $2^{\mathcal{O}(\sqrt{s}\log s)} n^{\mathcal{O}(1)}$.

4 Polynomial Time Algorithmic Results

In this section, we obtain polynomial-time algorithms for the optimization version of the Vertex k-Way Cut on interval graphs, circular-arc graphs, and permutation graphs.

4.1 Interval Graphs

Here, we design a dynamic-programming algorithm for the optimization version of the Vertex k-Way Cut on interval graphs. Let G be an interval graph with vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$. Since G is an interval graph, there exists a corresponding geometric intersection representation of G, where each vertex $v_i \in V(G)$ is associated with an interval $I_i = (\ell(I_i), r(I_i))$ in the real line, where $\ell(I_i)$ and $r(I_i)$ denote left and right endpoints, respectively in I_i. Two vertices v_i and v_j are adjacent in G if and only if their corresponding intervals I_i and I_j intersect with each other. Without loss of generality we can assume that along with the graph, we are also given the corresponding underlying intervals on the real line. We use $\mathcal{I}$ to denote the set $\{I_i \colon v_i \in V\}$ of intervals and P to denote the set of all endpoints of these intervals, i.e., $P = \cup_{I \in \mathcal{I}} \{\ell(I), r(I)\}$. In the remaining section, we use v_i and I_i interchangeably. For a pair of points a and b on the real line with $a \leq b$ (we say $a \leq b$ when x-coordinate of a is not greater than x-coordinate of b), we define $I_{a,b}$ to denote the intervals which are properly contained in $[a, b]$, formally $I_{a,b} = \{I \in \mathcal{I} \colon a \leq \ell(I) \leq r(I) \leq b\}$. Let $I_{\geqslant b}$ be the set of intervals whose left endpoints are greater than b and $I_{<b}$ be the set of intervals whose left endpoint is strictly less than b, formally $I_{\geqslant b} = \{I \in \mathcal{I} \colon \ell(I) \geq b\}$ and $I_{<b} = \{I \in \mathcal{I} \colon \ell(I) < b\}$.

We now define a table for dynamic-programming algorithm. For every tuple (i, x, y), where $1 \leq i \leq k$ and $x, y \in P$ with $x < y$, any cut where $G[I_{x,y}]$ is the i-th component with respect to the cut in $G[I_{<y}]$ is a feasible cut for the tuple (i, x, y) and $T[i; x, y]$ stores the minimum size among all such feasible cuts for the tuple (i, x, y). Notice that any two connected components do not intersect. Hence we can order the components from left to right. In particular, for a pair of components C_j and $C_{j'}$, we say $C_j \prec C_{j'}$ if for any pair of intervals $I \in C_j$ and $I' \in C_{j'}$ the condition $r(I) < \ell(I')$ holds. In the base case, we compute the values for $T[1; x, y]$ for each possible pair x, y in P where $x < y$. $T[1; x, y]$ stores the number of intervals in $G[I_{<y}]$ that have either left endpoint strictly less than x or right endpoint strictly greater than y, formally $T[1; x, y] = |I_{<y}| - |I_{x,y}|$.

In the next lemma, we give a recursive formula for computing the values $T[i; x, y]$ for $i > 1$.

Lemma 5. *For every integer i and every pair of points x, y in P where $2 \leq i \leq k$ and $x < y$, the following holds:*

$$T[i; x, y] = \min_{\substack{x', y' \in P \\ x' < y' < x}} \{T[i-1; x', y'] + |I_{<y} \cap I_{\geqslant y'}| - |I_{x,y}|\}.$$

Proof. We prove the recurrence relation by showing inequalities in both directions. In one direction, let $(C_1, C_2, \ldots, C_i)$ be a feasible cut corresponding to the entry $T[i; x, y]$. Here $C_i = G[I_{x,y}]$. Let x' and y' be the left endpoint and right endpoint of the component C_{i-1}, so $C_{i-1} \subseteq G[I_{x',y'}]$. Clearly, $x' < y' < x < y$. Now the intervals of the set $(I_{<y} \cap I_{\geqslant y'}) \setminus I_{x,y}$ are part of cut vertices corresponding to the entry $T[i; x, y]$. Here we can get a set of $(i-1)$ components $C_1, C_2, \ldots, C_{i-1}$ in the graph $G[I_{<y'}]$ with $C_{i-1} = G[I_{x',y'}]$ and cut of size at

most $T[i;x,y]-(|I_{<y}\cap I_{>y'}|-|I_{x,y}|)$. Therefore, by the definition of $T[i;x,y]$, $T[i-1;x',y']\leq T[i;x,y]-(|I_{<y}\cap I_{>y'}|-|I_{x,y}|)$.

In the other direction, let $(C'_1,C'_2,\ldots,C'_{i-1})$ be a feasible cut corresponding to the entry $T[i-1;x',y']$, where $x'<y'<x<y$ and $C_{i-1}=G[I_{x',y'}]$. Now the component induced by $I_{x,y}$ together with $C'_1,C'_2,\ldots,C'_{i-1}$ produces a feasible cut for $T[i;x,y]$. Therefore, the cut corresponding to $T[i-1;x',y']$ together with $(I_{<y}\cap I_{\geqslant y'})\setminus I_{x,y}$ gives a cut with the components $C'_1,\ldots,C'_{i-1},C'_i=G[I_{x,y}]$. Hence, $T[i-1;x',y']+|I_{<y}\cap I_{\geqslant y'}|-|I_{x,y}|\geq T[i;x,y]$. This completes the proof of the lemma. ◻

With the insight of Lemma 5, we can now state the following theorem.

Theorem 2. Vertex k-Way Cut *in interval graphs with n vertices can be solved in $\mathcal{O}(kn^4)$ time.*

Proof. Let G be a given graph with $\mathcal{I}$ as an interval representation where P denotes the set of endpoints of all the intervals. In the pre-processing step, we do the following: (i) for every point $p\in P$, we construct $I_{<p}$ and $I_{\geqslant p}$, (ii) for every pair of points p,q in P, we compute $|I_{p,q}|$ and $|I_{<p}\cap I_{\geqslant q}|$. It will take $\mathcal{O}(n^2)$ time to perform both these pre-processing steps. Now in the recurrence formula, to obtain $T[i;x,y]$, we use the already computed values $T[i;x',y']$ for each possible pair $x',y'\in P$ with $x'<y'<x<y$. Computing any entry takes $\mathcal{O}(n^2)$ time. Since i ranges from 1 to k, we can compute all the values $T[i;x,y]$ in $\mathcal{O}(kn^4)$ time. Notice that the entry $T[k;.,.]$ with minimum value gives us the size of a minimum vertex k-way cut in G. Hence, the theorem holds. ◻

4.2 Proper Interval Graphs

In this subsection, we design a dynamic-programming algorithm for the optimization version of the Vertex k-Way Cut on proper interval graphs. In proper interval graphs, each vertex is associated with an interval in the real line such that no interval is completely contained in another interval. We use the notations $\mathcal{I}$, I_i, $\ell(I_i)$, $r(I_i)$ and P with the same definitions as used in the previous subsection. Let $\mathcal{I}$ be the set of all intervals with ordering $I_1<I_2<\ldots<I_n$ according to their left endpoints. Observe that for proper interval graphs, the ordering of intervals with respect to their left endpoints is same as with respect to their right endpoints. More explicitly, for any two intervals I_i and I_j where $\ell(I_i)<\ell(I_j)$, $r(I_i)$ must be less than $r(I_j)$. Let $\mathcal{I}_i=\{I_1,I_2,\ldots,I_i\}$ and $G[\mathcal{I}_i]$ denotes the subgraph of G induced by $\mathcal{I}_i$. Also for an interval I_i, I_i^ℓ denotes the interval in $\mathcal{I}$ which has leftmost left endpoint among all the intervals containing $\ell(I_i)$, formally, $I_i^\ell=I_c$, where $c=\min\{j;I_j\in\mathcal{I},\ell(I_j)<\ell(i)<r(I_j)\}$.

We now define a table for dynamic-programming algorithm. For every pair (i,t), where $1\leq i\leq n$ and $1\leq t\leq k$, we define two entries. $T[\in;\ i,\ t]$ and $T[\notin;\ i,\ t]$. For every tuple $(\in,i,t)$, any cut where the interval I_i lies in one of the t components with respect to the cut in $G[\mathcal{I}_i]$ is a feasible cut for the tuple $(\in,i,t)$ and $T[\in;\ i,\ t]$ stores the minimum size among all such feasible cuts for the tuple $(\in, i, t)$. For every tuple $(\notin, i, t)$, any cut where the interval I_i does not lie in any of the t components with respect to the cut in $G[\mathcal{I}_i]$ is a feasible cut for the tuple $(\notin, i, t)$ and $T[\notin;\ i,\ t]$ stores the minimum size among all such feasible cuts for the tuple $(\notin, i, t)$. Similar to interval graphs, here also we order the components from left to right. In particular, for a pair of components C_j and $C_{j'}$, we say $C_j \prec C_{j'}$ if for any pair of intervals $I \in C_j$ and $I' \in C_{j'}$ the condition $r(I) < \ell(I')$ holds.

In the base case, the values $T[\in;\ i,\ 1] = 0$ and $T[\notin;\ i,\ 1] = 1$, for $i \in [n]$.

In the next two lemmas, we give recursive formulas for computing the values $T[\in;\ i,\ t]$ and $T[\notin;\ i,\ t]$, for $i \in [n],\ 1 < t \leq k$.

Lemma 6. *For every t and i where $2 \leq t \leq k$ and $1 \leq i < n$, the following holds:*

$$T[\notin;\ i+1,\ t] = 1 + \min\{T[\in;\ i,\ t],\ T[\notin;\ i,\ t]\}.$$

Proof. We prove the given recurrence by showing inequalities in both directions. In one direction, let $(C_1, C_2, \ldots, C_t)$ be a feasible cut corresponding to the entry $T[\notin;\ i+1,\ t]$. We distinguish the following two cases. Case 1: If $I_i \in C_t$, then $(C_1, C_2, \ldots, C_t)$ is a feasible cut corresponding to the entry $T[\in;\ i,\ t]$. Case 2: If $I_i \notin C_t$ then $(C_1, C_2, \ldots, C_t)$ is a feasible cut corresponding to the entry $T[\notin;\ i,\ t]$. In both these cases, the cut size is one less than a cut corresponding to $T[\notin;\ i+1,\ t]$. Therefore, $T[\notin;\ i+1,\ t] - 1 \geq \min\{T[\in;\ i,\ t],\ T[\notin;\ i,\ t]\}$.

In the other direction, let $(C'_1, C'_2, \ldots, C'_t)$ be a feasible cut respecting the tuple $(\in, i, t)$, where X_1 is the corresponding set of cut vertices. Now $(C'_1, C'_2, \ldots, C'_t)$ is also a feasible cut for $T[\notin;\ i+1,\ t]$ with $X_1 \cup \{I_{i+1}\}$ considered as the set of cut vertices. Similarly, let $(C''_1, C''_2, \ldots, C''_t)$ be a feasible cut corresponding to the entry $T[\notin;\ i,\ t]$, where X_2 is a set of cut vertices. Now $(C''_1, C''_2, \ldots, C''_t)$ is also a feasible cut corresponding to the entry $T[\notin;\ i+1,\ t]$ where $X_2 \cup \{I_{i+1}\}$ is a set of cut vertices. Thus, $T[\notin;\ i+1,\ t] \leq 1 + \min\{T[\in;\ i,\ t],\ T[\notin;\ i,\ t]\}$. Hence the lemma holds. ☐

Lemma 7. *Let d_i be the number of intervals passing through $\ell(I_i)$ and i' be the index corresponding to the interval I_i^ℓ. Then for every $2 \leq t \leq k$ the following holds:*

$$T[\in;\ i+1,\ t] = \min\{T[\in;\ i,\ t],\ T[\notin;\ i',\ t-1] + d_{i+1} - 1\}.$$

Proof. We prove the recurrence relation by showing inequalities in both directions. In one direction, let $(C_1, C_2, \ldots, C_t)$ be a feasible cut corresponding to the entry $T[\in;\ i+1,\ t]$. We distinguish the following two cases. If $I_i \in C_t$ then $(C_1, C_2, \ldots, (C_t \setminus \{I_{i+1}\}))$ is a feasible cut corresponding to the entry $T[\in;\ i,\ t]$. If $I_i \notin C_t$, then $(C_1, C_2, \ldots, C_{t-1})$ is a feasible cut corresponding to the entry $T[\notin;\ i',\ t-1]$, but in this case the cut size decreases by $d_{i+1} - 1$. So $T[\in;\ i+1,\ t] \geq \min\{T[\in;\ i,\ t],\ T[\notin;\ i',\ t-1] + d_{i+1} - 1\}$. In the other direction, let $(C'_1, C'_2, \ldots, C'_t)$ be a feasible cut corresponding to the entry $T[\in;\ i,\ t]$, where X_1 is the set of cut vertices. Now $(C'_1, C'_2, \ldots, C'_t \cup \{I_{i+1}\})$ is also a feasible cut corresponding to the entry $T[\in;\ i+1,\ t]$ with the same cut X_1. Similarly, let

$(C''_1, C''_2, \ldots, C''_{t-1})$ be a feasible cut corresponding to the entry $T[\notin;\ i',\ t-1]$, where X_2 is the set of cut vertices. Let Z denote the set of intervals containing $\ell(I_{i+1})$ except I_{i+1}. Now $(C''_1, C''_2, \ldots, C''_{t-1}, I_{i+1})$ is also a feasible cut corresponding to the entry $T[\in;\ i+1,\ t]$ with $X_2 \cup Z$ as a set of cut vertices. Since $|Z| = d_{i+1}$, then $T[\in;\ i+1,\ t] \leq \min\{T[\in;\ i,\ t],\ T[\notin;\ i',\ t-1] + d_{i+1} - 1\}$. ◻

With the insight of Lemma 6 and Lemma 7, we can now state the following theorem.

Theorem 3. Vertex k-Way Cut *in proper interval graph with n vertices can be solved in $\mathcal{O}(kn)$ time assuming that the interval model is given..*

Proof. Let G be a given proper interval graph with corresponding set $\mathcal{I}$ of n intervals. Let P denote the set of all endpoints of these intervals. Here we assume that we are given the set of intervals with the ordering based on left endpoints as an input. In the pre-processing step, we do the following: compute I_i^ℓ and d_i, for each interval $I_i \in \mathcal{I}$. It will take $\mathcal{O}(n)$ time to perform all the pre-processing steps. Now in the recurrence formula, to obtain $T[\notin;\ i+1,\ t]$ and $T[\in;\ i+1,\ t]$, we use $\mathcal{O}(1)$ many computations. So computing any entry takes $\mathcal{O}(1)$ time. Since i ranges from 1 to up to n, and $t \leq k$, we can compute all the entries of the table in $\mathcal{O}(kn)$ time. Notice that the entry $T[.; n, k]$ with minimum value gives us the size of a minimum vertex k-way cut in G. Hence, the theorem holds. ◻

4.3 Circular-arc Graphs

A graph G is said to be a circular-arc graph if there exists a corresponding geometric intersection representation $\mathcal{A}(G)$ of G, where each vertex $v \in G$ is associated with an arc on a fixed circle. Two vertices u and v are adjacent in G if and only if the corresponding arcs intersect each other. It is easy to observe that this graph class contains interval graphs.

Here we design a polynomial-time algorithm for the optimization version of Vertex k-Way Cut problem on circular-arc graphs. Let S be an optimal solution of Vertex k-Way Cut problem on G and C be a component in $G \setminus S$. Assume I is the circular-arc representation of C in $\mathcal{A}(G)$ and $I_1 \in I$ be the arc that has the last endpoint, say u, in the clockwise direction in the circular-arc representation of $G \setminus S$. Let I' be the set of arcs in $\mathcal{A}(G)$ that intersect u, excluding I_1. Since S is a k-way cut it must contain all the vertices corresponding to the arcs in I'. Now assume we cut the circle corresponding to the circular-arc representation of $G \setminus S$ at u and convert the circular-arc to a real line to get an instance of Vertex k-Way Cut problem on interval graphs. We claim that $S \setminus I'$ is an optimal solution to the Vertex k-Way Cut problem on the interval graph instance that we construct.

Claim. $S \setminus I'$ is a solution to the Vertex k-Way Cut problem on the interval graph instance $G \setminus I'$.

Proof. Let S' be an optimal solution on the Vertex k-Way Cut problem on the interval graph induced by $G \setminus I'$. If $|S'| = |S \setminus I'|$, we are done. Else, $|S'| < |S \setminus I'|$

then $S \setminus I'$ is not an optimal solution to the VERTEX k-WAY CUT problem on the interval graph instance $G \setminus I'$. Observe that $G \setminus (S' \cup I')$ has at least k components, and $|S' \cup I'| = |S'| + |I'| < |S| + |I'| = |S \cup I'|$. Thus $S' \cup I'$ is an optimal solution to VERTEX k-WAY CUT problem on G with size strictly smaller than S which is a contradiction to our assumption that S is an optimal solution. □

Now given an instance G for VERTEX k-WAY CUT problem on circular-arc graphs we convert it to an instance of interval graph for all the $2n$ endpoints and run the algorithm for VERTEX k-WAY CUT problem, designed in Section 4.1, on each of those interval graphs and store the corresponding S', I'. As a solution, we return the set $S' \cup I'$ that has minimum size. Since algorithm for interval graph runs in $\mathcal{O}(kn^4)$ time (Theorem 2); so we have the following theorem.

Theorem 4. VERTEX k-WAY CUT *in circular-arc graphs with n vertices can be solved in $\mathcal{O}(kn^5)$ time.*

4.4 Permutation Graphs

This subsection presents a dynamic-programming algorithm for the optimization version of the VERTEX k-WAY CUT problem on permutation graphs. Let G be a permutation graph with vertex set $V(G)$ and edge set $E(G)$. There exists a corresponding geometric intersection representation for a permutation graph G similar to interval graphs, where each vertex v in G is associated with a line segment $S(v)$ with endpoints $x(v)$ and $y(v)$ being on two parallel lines X and Y, respectively. Without loss of generality, we can assume that both the lines X and Y are horizontal. Two vertices u and v are adjacent in G if and only if the segments $S(u)$ and $S(v)$ intersect with each other. Assume that along with the graph, we have the set of corresponding line segments as an input. Here, we use $\mathcal{S}$ to denote the segments $\{S(v) \colon v \in V\}$. Let P_X and P_Y denote the set of all endpoints of S on the lines X and Y, respectively. Let $P = P_X \cup P_Y$.

For a pair of vertices u and v, we write $x(u) < x(v)$ (similarly, $y(u) < y(v)$) to indicate that $x(v)$ is to the right of $x(u)$ (similarly, $y(v)$ is to the right of $y(u)$). If both $x(u) < x(v)$ and $y(u) < y(v)$ hold, then we say $S(u) \prec S(v)$. In the rest of this subsection, we interchangeably use v and $S(v)$. For a pair of points α and β where $\alpha \in X, \beta \in Y$, we denote the set of segments in $\mathcal{S}$ whose one endpoint lies either to the left of α or to the left of β by S^{α}_{β}. We use $G[\alpha, \beta]$ to denote the subgraph induced by S^{α}_{β} in G. Additionally, for any set of four points, $\alpha_1, \alpha_2 \in X$ and $\beta_1, \beta_2 \in Y$ such that $\alpha_1 < \alpha_2$ and $\beta_1 < \beta_2$, we define $S^{\alpha_1,\alpha_2}_{\beta_1,\beta_2} = \{S(v) \colon \alpha_1 \le x(v) \le \alpha_2,\ \beta_1 \le y(v) \le \beta_2\}$. We use $G[(\alpha_1, \alpha_2), (\beta_1, \beta_2)]$ to denote the subgraph of G induced by the segments $S^{\alpha_1,\alpha_2}_{\beta_1,\beta_2}$.

We now define a table for our dynamic-programming algorithm. For every tuple (i, p, q, r, s), where $p, q \in P_X$ with $p < q$ and $r, s \in P_Y$ with $r < s$, any cut where $G[(p, q), (r, s)]$ is the i-th component with respect to the cut in $G[q, s]$ is a feasible cut for the tuple (i, p, q, r, s) and $T[i; p, q, r, s]$ stores the minimum size among all such feasible cut for the tuple (i, p, q, r, s). Notice that any two connected components do not intersect. Hence we can order the components from left to right. In particular, for a pair of components C_j and $C_{j'}$, we say $C_j \prec C_{j'}$ if for any pair of line segments $u \in C_j$ and $v \in C_{j'}$, $S(u) \prec S(v)$.

For the base case, the value $T[1; p, q, r, s]$ is the number of segments in $G[q, s]$ whose one endpoint lies either strictly to the left of p or r, or strictly to the right of q or s, formally $T[1; p, q, r, s] = |S^q_s| - |S^{p,q}_{r,s}|$. In the next lemma, we give a recursive formula for computing the values $T[i; p, q, r, s]$, for $i > 1$.

Lemma 8. *For every i, $2 < i < k$ and any set of four points p, q, r, s, where $p, q \in P_X$ with $p < q$ and $r, s \in P_Y$ with $r < s$, the following holds:*

$$T[i; p, q, r, s] = \min_{\substack{p', q' \in P_X \ \&\ r', s' \in P_Y \\ p' < q' < p,\ r' < s' < r}} \{T[i-1; p', q', r', s'] + |S^q_s| - |S^{q'}_{s'}| - |S^{p,q}_{r,s}|\}.$$

Proof. We prove the recurrence by showing inequalities in both directions. In one direction, let $(C_1, C_2, \ldots, C_i)$ be a feasible cut corresponding to the entry $T[i; p, q, r, s]$. Here $C_i = G[(p, q), (r, s)]$. Let p', q', r', s' be four points such that $C_{i-1} = G[(p', q'), (r', s')]$, $p', q' \in P_X$ and $r', s' \in P_Y$. Clearly, $p' < q' < p$ and $r' < s' < r$ hold. Now, the segments of the set $S^q_s \setminus (S^{q'}_{s'} \cup S^{p,q}_{r,s})$ are cut vertices corresponding to the entry $T[i; p, q, r, s]$. Here we get a set of $(i-1)$ components $C_1, C_2, \ldots, C_{i-1}$ in the graph $G[q', s']$ with $C_{i-1} \subseteq G[(p', q'), (r', s')]$ and cut size at most $T[i; p, q, r, s] - (|S^q_s| - |S^{q'}_{s'}| - |S^{p,q}_{r,s}|)$. Therefore, $T[i-1; p', q', r', s'] \leq T[i; p, q, r, s] - (|S^q_s| - |S^{q'}_{s'}| - |S^{p,q}_{r,s}|)$.

In the other direction, let $(C'_1, C'_2, \ldots, C'_{i-1})$ be a feasible cut corresponding to the entry $T[i-1; p', q', r', s']$, where $p' < q' < p$, $r' < s' < r$ and $C_{i-1} = G[(p', q'), (r', s')]$. The component induced by the subgraph $G[(p, q), (r, s)]$ together with $C'_1, C'_2, \ldots, C'_{i-1}$ produces a feasible cut for $T[i; p, q, r, s]$. Now the cut corresponding to the entry $T[i-1; p', q', r', s']$ together with $(|S^q_s| - |S^{q'}_{s'}| - |S^{p,q}_{r,s}|)$ gives a cut that yields the set of components $C'_1, C'_2, \ldots, C'_{i-1}$, $C'_i = G[(p, q), (r, s)]$. Hence, $T[i-1; p', q', r', s'] + |S^q_s| - |S^{q'}_{s'}| - |S^{p,q}_{r,s}| \geq T[i; p, q, r, s]$. This completes the proof of the lemma. ◻

With the insight of Lemma 8, we can now state the following theorem.

Theorem 5. Vertex k-Way Cut *in permutation graph with n vertices can be solved in $\mathcal{O}(kn^8)$ time.*

Proof. Let G be a given graph with a set $\mathcal{S}$ of n line segments. Recall that we use P_X and P_Y to denote the set of all endpoints of line segments in X and Y, respectively. In the pre-processing step, we do the following: (i) we construct S^α_β, for every pair of points $\alpha \in P_X$ and $\beta \in P_Y$. (ii) we compute $|S^{\alpha_1,\alpha_2}_{\beta_1,\beta_2}|$ for each possible set of four points $\alpha_1, \alpha_2 \in P_X$ and $\beta_1, \beta_2 \in P_Y$. It takes $\mathcal{O}(n^5)$ time to perform all these pre-processing steps. Now in the recurrence formula, to obtain $T[i; p, q, r, s]$, we use the already computed values, where $p', q' \in P_X$ and $r', s' \in P_Y$ with $p' < q' < p$ and $r' < s' < r$. Computing any entry takes $\mathcal{O}(n^4)$ time. Since i ranges from 1 to k, we can compute all the values $T[i; p, q, r, s]$ in $\mathcal{O}(kn^8)$ time. Notice that the entry $T[k; ., .]$ with minimum value gives us the size of a minimum vertex k-way cut in G. Hence, the theorem holds. ◻

References

1. Christopher J Augeri and Hesham H Ali. New graph-based algorithms for partitioning VLSI circuits. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No. 04CH37512)*, volume 4, pages IV–IV. IEEE, 2004.
2. Chandra Chekuri, Kent Quanrud, and Chao Xu. LP relaxation and tree packing for minimum k-cut. *SIAM Journal on Discrete Mathematics*, 34(2):1334–1353, 2020.
3. Derek G Corneil. The complexity of generalized clique packing. *Discrete applied mathematics*, 12(3):233–239, 1985.
4. Marek Cygan, Pawel Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Saket Saurabh, and Magnus Wahlström. Randomized Contractions Meet Lean Decompositions. *ACM Trans. Algorithms*, 17(1):6:1–6:30, 2021.
5. Rodney G Downey, Vladimir Estivill-Castro, Michael Fellows, Elena Prieto, and Frances A Rosamund. Cutting up is hard to do: The parameterised complexity of k-cut and related problems. *Electronic Notes in Theoretical Computer Science*, 78:209–222, 2003.
6. Olivier Goldschmidt and Dorit S Hochbaum. A polynomial algorithm for the k-cut problem for fixed k. *Mathematics of operations research*, 19(1):24–37, 1994.
7. Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs.* Elsevier, 2004.
8. Anupam Gupta, Euiwoong Lee, and Jason Li. An FPT algorithm beating 2-approximation for k-cut. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2821–2837. SIAM, 2018.
9. Anupam Gupta, Euiwoong Lee, and Jason Li. Faster exact and approximate algorithms for k-cut. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 113–123. IEEE, 2018.
10. David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.
11. Ken-ichi Kawarabayashi and Bingkai Lin. A nearly 5/3-approximation FPT Algorithm for Min-k-Cut. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 990–999. SIAM, 2020.
12. Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 160–169. IEEE, 2011.
13. Jason Li. Faster minimum k-cut of a simple graph. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1056–1077. IEEE, 2019.
14. Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A Parameterized Approximation Scheme for Min k-Cut. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 798–809. IEEE, 2020.
15. Pasin Manurangsi. Inapproximability of maximum edge biclique, maximum balanced biclique and minimum k-cut from the small set expansion hypothesis. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
16. Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.
17. Huzur Saran and Vijay V Vazirani. Finding k cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–108, 1995.
18. Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.