A Contraction Tree SAT Encoding for Computing Twin-Width

Yinon Horev[1], Shiraz Shay[1], Sarel Cohen[1], Tobias Friedrich[2], Davis Issac[2], Lior Kamma[1], Aikaterini Niklanovits[2]*, and Kirill Simonov[2]

[1] The Academic college of Tel Aviv-Yaffo, Israel.
{yinonho,shirazsh,sarelco,liorkm}@mta.ac.il
[2] Hasso Plattner Institute, University of Potsdam, Germany.
{tobias.friedrich,davis.issac,aikaterini.niklanovits,kirill.simonov}@hpi.de

Abstract. Twin-width is a structural width parameter and matrix invariant introduced by Bonnet et al. [FOCS 2020], that has been gaining attention due to its various fields of applications. In this paper, inspired by the SAT approach of Schidler and Szeider [ALENEX 2022], we provide a new SAT encoding for computing twin-width. The encoding aims to encode the contraction sequence as a binary tree. The asymptotic size of the formula under our encoding is smaller than in the state-of-the-art relative encoding of Schidler and Szeider. We also conduct an experimental study, comparing the performance of the new encoding and the relative encoding.

Keywords: Twin-width · SAT encoding.

1 Introduction

Twin-width is a graph and matrix invariant recently introduced by Bonnet et al [7], [4], [5]), inspired by a width invariant defined on permutations by Guillemot and Marx [9]. Since its inception, twin-width received tremendous interest in the scientific community. From the algorithmic perspective, the benefits of twin-width are twofold. First, many diverse graph families are known to have bounded twin-width, for example graphs of bounded treewidth or clique-width, graphs excluding a fixed minor, planar graphs, posets of bounded width (in particular, unit interval graphs) [7]. Second, many NP-hard problems are solvable in polynomial time on graphs of bounded twin-width.

The latter property is formalized by Bonnet et al [7] as follows: Given an n-vertex graph G, a witness that its twin-width is at most d, and a first-order sentence ϕ, it can be decided whether ϕ holds on G in $f(d, \phi)n$ time, where f is some computable function. It is worth noting, that this result does not give directly algorithms with practical running times since f is an extremely fast-growing function; this is a common drawback of algorithmic meta-theorems. However, several important NP-hard problems that are expressible by a first-order sentence

* Corresponding author.

of bounded size, are also known to be solvable on graphs of bounded twin-width directly via dynamic programming with efficient running times; in particular, k-Clique, k-Dominating Set, and k-Ones SAT for bounded k [5],[8].

Given the above, the natural question is, whether the twin-width of a graph can be computed exactly. That is, given a graph G, can we find the smallest d such that d-contraction sequence exists? Unfortunately, in the general case this turns out to be intractable: even deciding whether a graph has twin-width 4 is NP-complete [2]. While for many graph classes twin-width is bounded, very few results are known for computing (or approximating) twin-width even when the given graph comes from a special graph class. For example, Král and Lamaison [11] showed that planar graphs have twin-width at most 8; however it is wide open whether we can compute the twin-width of a *given* planar graph (since it may also be smaller than 8). This motivates turning our attention to heuristic methods of computing twin-width. The high demand for such results is also illustrated by the 2023 edition of the PACE challenge[3], which focuses exclusively on computing twin-width. For the purpose of this work we only consider algorithms that yield a provably optimal contraction sequence; however, the running time is not necessarily bounded by a polynomial in n in general.

This line of research was pioneered by Schidler and Szeider [14], who devised a SAT encoding for computing twin-width. By supplying that encoding to a SAT solver, they were able to identify the exact value of twin-width for a variety of named graphs. In fact, they presented two different SAT encodings called *absolute* and *relative*. Interestingly, in all their tests the relative encoding vastly outperforms the absolute encoding, despite the fact that the formula in the relative encoding is larger: $\mathcal{O}(n^4)$ clauses versus $\mathcal{O}(n^3)$ clauses for the absolute encoding for an n-vertex graph. To the best of our knowledge, no other SAT encodings for computing twin-width were studied so far.

Our contribution. In this work we propose an alternative SAT encoding for computing twin-width, which is conceptually different from the encodings of Schidler ans Szeider [14]. We prove formally the correctness of the encoding, and argue that the size of the formula in the encoding is only $\mathcal{O}(n^3)$ clauses, which is a asymptotically smaller than $\Omega(n^4)$ in the relative encoding of [14]. We also supply an implementation of the encoding, and conduct empirical tests comparing the performance of our encoding to that of the state-of-the-art relative encoding. The results show that there are cases where the new encoding allows to compute twin-width much faster, although the converse happens as well. We highlight that our main contribution is presenting a new SAT-encoding based on a binary contraction tree that is significantly different than the encoding presented so far. In the context of our theoretical analysis, we prove the correctness of our SAT-encoding and analyze the big-O size of the formula, while the experimental part is mainly to empirically validate the correctness and feasibility of the encoding.

[3] PACE stands for Parameterized Algorithms and Computational Experiments; the challenge is dedicated to bringing the gap between theoretical and practical parameterized algorithms. The official website of the challenge is https://pacechallenge.org/2023/.

Related work. Here we list some of the known results on twin-width. Graphs of twin-width 0 are exactly cographs, and can be recognized in poly-time [7]. Later it was shown that graphs of twin-width 1 can also be recognized in poly-time [6]. Jacob and Pilipczuk [10] show, among other results, that twin-width of a graph is at most $3 \cdot 2^{\mathrm{tw}-1}$, where tw is the treewidth of the graph; it is also known that twin-width can be exponential in treewidth [3]. Balabán and Hliněný [1] show that twin-width is linear in the poset width, which implies that twin-width of unit interval graphs is at most two, and can be computed in poly-time. Twin-width of planar graphs is at most 8 [11], and can be as high as 7 [12].

2 Preliminaries

All graphs mentioned in this paper are simple, undirected and finite and we use standard graph-theoretic notations. In particular, given a graph G, we denote by $V(G)$ and $E(G)$ its set of vertices (or nodes) and edges respectively. Moreover, given a vertex set $S \subseteq V(G)$ we denote by $G[S]$ and $G-S$, the graph induced by the vertices of S and the graph induced by the vertices $V(G) - S$ respectively. When referring to the *open neighborhood* of a vertex $v \in V(G)$, i.e. the set of neighbors of v without v, we write $N_G(v)$, while we omit G when the graph we refer to is clear from the context. Similarly, we denote by $N_G[v]$ the *closed neighborhood* of v, i.e. $N_G(v) \cup v$. Two distinct vertices u, v are called *false twins* if $N(u) = N(v)$ and *true twins* if $N[u] = N[v]$. Given a pair $u, v \in V(G)$ we characterize the process of deleting those vertices and creating a new one with neighborhood $N(u) \cup N(v)$ as *contraction of* u, v. The graph that occurs from G after the contraction of u, v is denoted by $G/u, v$.

We now proceed in defining twin-width, following the definition of Bonnet et al in [7]: A graph $G = (V, B, R)$ is a *trigraph* if B and R are two disjoint sets of edges on V (referred to as black and red respectively). Note that an ordinary graph is a trigraph where $B = E(G)$ and $R = \emptyset$. A trigraph (V, B, R) such that (V, R) has maximum degree d is called a d-trigraph. The neighborhood of a vertex v on a trigraph is all of its adjacent vertices, regardless of the color of the edge that connects them. Given a trigraph $G = (V, B, R)$ and a pair of distinct vertices $u, v \in V(G)$ we define the trigraph $G' = G/u, v$ such that, for the neighbors of the vertex w that occur from the contraction the following holds: A vertex $x \in N(w)$ is connected to w through a black edge if and only if it was connected through a black edge to both u, v in G. Otherwise (if at least one was already connected through a red edge or if x was not adjacent to both of the contracted vertices), the edge connecting x to w in G' is red.

A sequence of d-contractions for a trigraph is a sequence of d-trigraphs $G_0, G_1, \ldots, G_{n-1}$, where $G_0 = G$, $|V(G_{n-1})| = 1$ and G_i for $i \geq 1$ is obtained from G_{i-1} by a contraction (see Figure 1 for an example). The twin-width of a graph is the smallest d for which a d-sequence exists and is denoted by $tww(G)$. Such a sequence is called a *d-contraction sequence.*

Since the main result of this paper is based on encoding a contraction sequence as a binary tree, we also define what the Conjunctive Normal Form (CNF) of a formula is. A literal is a (propositional) variable or the negation of it, and we call a clause a disjunction of literals. A formula ϕ is in Conjunctive Normal Form (CNF) if it is the conjunction of clauses.

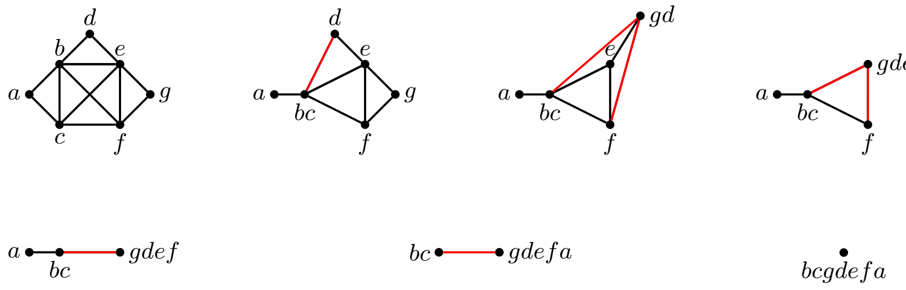

Fig. 1. A 2-contraction sequence of a graph.

3 Binary SAT Encoding

To obtain our binary SAT encoding of the d-contraction sequence problem we express all restrictions into propositional logic and then convert them to CNF. Throughout the description of our encoding we refer to $V(G)$ as vertices, and to $V(T)$ as nodes.

Let $G = (V, E)$ be the input graph and consider an initialization of E, denoting by $\neg edge_{i,j}$ the non-existing edges between vertices i, j, by $edge_{i,j}$ the existing ones, and by $\neg red_{i,j}$ the absence of red edges on the input graph.

$$\left(\bigwedge_{\substack{i,j\in[n],j>i \\ ij\notin E}} \neg edge_{i,j} \right) \wedge \left(\bigwedge_{\substack{i,j\in[n],j>i \\ ij\in E}} edge_{i,j} \right) \wedge \left(\bigwedge_{i,j\in[n],j>i} \neg red_{i,j} \right)$$

Observe that a contraction sequence can be represented by a rooted binary tree, where the leaves correspond to the vertices of G, internal nodes to subsequent contractions, and the root corresponds to the final contraction. Since twin-width is always at most d when the number of vertices does not exceed $d + 1$, we can avoid encoding the final $d + 1$ contractions in the sequence. In terms of the tree representation, this corresponds to removing the nodes of the tree representing the final $d + 1$ contractions, which results in a binary forest with at most $d+2$ trees. We refer to the binary forest of the sought-after optimal contraction sequence as T. We note that $|V(T)| \leq 2n - d - 2$ since the whole contraction tree contains $2n - 1$ nodes as a binary tree with $n = |V(G)|$ leaves, and the nodes corresponding to the final $d + 1$ contractions are not considered.

To encode T we define variables $lc_{i,j}$ and $rc_{i,j}$ for each pair of vertices, which are true if and only if j is the left child of i and j is the right child of i respectively. Using these variables we ensure that T is binary by encoding the following:

- Each node has at most one parent.

$$\bigwedge_{i\in[2n-d-3]} \left\langle \sum_{j=max(i+1,n+1)}^{2n-d-2} (lc_{j,i} + rc_{j,i}) \leq 1 \right\rangle$$

  Note that the final vertex is not included in this encoding.
- Each parent node has exactly one left and one right child.

$$\bigwedge_{j\in[n+1,2n-d-2]} \left\langle \sum_{i\in[j]} rc_{j,i} = 1 \right\rangle \qquad \bigwedge_{j\in[n+1,2n-d-2]} \left\langle \sum_{i\in[j]} lc_{j,i} = 1 \right\rangle$$

Lemma 1. *The encoding above produces a binary forest on $2n-d-2$ nodes.*

Proof. Assume that there is a node v_i that has at least two parents v_x, v_y, where $x, y > i$. Then the clause $\bigwedge_{i\in[2n-d-3]} \left\langle \sum_{j=max(i+1,n+1)}^{2n-d-2} (lc_{j,i} + rc_{j,i}) \leq 1 \right\rangle$ corresponding to v_i is false since $\sum_{j=max(i+1,n+1)}^{2n-d-2} (lc_{j,i} + rc_{j,i})$ includes $(lc_{x,i} + rc_{x,i}) + (lc_{y,i} + rc_{y,i})$ which is equal to 2, leading to contradiction. Note that we construct those clauses only considering values greater than n, since all the vertices of G correspond to leaf nodes.

Regarding the fact that each parent has exactly one left and one right child, we do not consider the leaves and hence we start from $n+1$. Here it suffices to note that for each possible parent node we check all the possible right and left children to ensure that it has exactly one of each.

We proceed with encoding the already contracted vertices through the variable $vanish_{p,i}$. In particular, we define this variable to be true if i is contracted to any vertex with number at most p. Intuitively, this variable is true if and only if at the time when the node p is formed through some contraction, the node i doesn't exist any more. To implement the semantics we encode the following:

- Each leaf node is vanished at the moment its parent node is formed and hence, for each $i \in [n]$

$$vanish_{n+1,i} \iff lc_{n+1,i} \vee rc_{n+1,i}$$

- Iteratively defined, an internal node i is already vanished when p is formed, either if it is one of its children or if it has been vanished in some previous contraction, due to it being a child of some other contracted node. Hence, for each $p \in [n+2, 2n-d-2]$, $i \in [p-1]$

$$vanish_{p,i} \iff lc_{p,i} \vee rc_{p,i} \vee vanish_{p-1,i}$$

- Lastly, in order to ensure that no node is considered to be "contracted to itself" at the time it is formed, we set for each $p \in [n+1, 2n-d-2]$

$$\neg vanish_{p,p}$$

We now encode the edges of G between the children of some node and the other nodes. In particular, the left adjacency of i, p, denoted $la_{i,p}$, means that there is an edge between the vertex represented by the left child of p and the one represented by i. Similarly the right adjacency of i, p is defined, and denoted by $ra_{i,p}$. In the list encoding these adjacencies below, the first argument refers to the child of p while the second denotes whether or not an edge (red or black) to vertex c exists. The minimum value is placed first when choosing the edge, to be consistent with how we initially encoded $E(G)$. For each $p \in [n+1, 2n-d-2]$, $c \in [p-1]$, $i \in [p-1]$ such that $i \neq c$, and similarly for their negation

$$\begin{array}{ll} lc_{p,c} \wedge edge_{min(i,c),max(i,c)} \Rightarrow la_{i,p}, & rc_{p,c} \wedge edge_{min(i,c),max(i,c)} \Rightarrow ra_{i,p} \\ lc_{p,c} \wedge red_{min(i,c),max(i,c)} \Rightarrow lr_{i,p}, & rc_{p,c} \wedge red_{min(i,c),max(i,c)} \Rightarrow rr_{i,p} \\ lc_{p,c} \wedge \neg edge_{min(i,c),max(i,c)} \Rightarrow \neg la_{i,p}, & rc_{p,c} \wedge \neg edge_{min(i,c),max(i,c)} \Rightarrow \neg ra_{i,p} \\ lc_{p,c} \wedge \neg red_{min(i,c),max(i,c)} \Rightarrow \neg lr_{i,p}, & rc_{p,c} \wedge \neg red_{min(i,c),max(i,c)} \Rightarrow \neg rr_{i,p} \end{array}$$

Using the right and left adjacencies of each node, we are able to encode the edges created from the contractions. In particular, an edge connecting node i that occurs from some contraction to another node j exists, if any of the children of i (the ones that get contracted in order to create i) is adjacent in G to j, and j still exists at the moment i is created. Moreover, this edge is red if exactly one of those children is adjacent to j. Formally, this is encoded as follows:

For each $i \in [n+1, 2n-d-2]$, $j \in [i-1]$

$$\begin{aligned} edge_{j,i} &\iff (la_{j,i} \vee ra_{j,i}) \wedge \neg vanish_{i,j} \\ red_{j,i} &\iff edge_{j,i} \wedge (lr_{j,i} \vee rr_{j,i} \vee (la_{j,i} \oplus ra_{j,i})) \end{aligned}$$

The encoding of the edges occurring from the contraction we described above $edge_{j,i} \iff (la_{j,i} \vee ra_{j,i}) \wedge \neg vanish_{i,j}$ is converted to CNF as follows:

$$(\neg edge_{j,i} \vee la_{j,i} \vee ra_{j,i}) \wedge (\neg vanish_{i,j} \vee \neg edge_{j,i}) \wedge (\neg la_{j,i} \vee edge_{j,i} \vee vanish_{i,j}) \\ \wedge(\neg ra_{j,i} \vee edge_{j,i} \vee vanish_{i,j}))$$

Similarly for the encoding of the red edges, $red_{j,i} \iff edge_{j,i} \wedge (lr_{j,i} \vee rr_{j,i} \vee (la_{j,i} \oplus ra_{j,i}))$ we have:

$$red_{j,i} \iff edge_{j,i} \wedge (lr_{j,i} \vee rr_{j,i} \vee ((la_{j,i} \vee ra_{j,i}) \wedge (\neg la_{j,i} \vee \neg ra_{j,i})))$$

$$\begin{aligned} red_{j,i} \iff (edge_{j,i}) \wedge (lr_{j,i} \vee rr_{j,i} \vee la_{j,i} \vee ra_{j,i}) \\ \wedge(lr_{j,i} \vee rr_{j,i} \vee \neg la_{j,i} \vee \neg ra_{j,i}) \end{aligned}$$

$$\begin{aligned} &\Rightarrow (\neg red_{j,i} \vee edge_{j,i}) \wedge (\neg red_{j,i} \vee lr_{j,i} \vee rr_{j,i} \vee la_{j,i} \vee ra_{j,i}) \\ &\quad \wedge (\neg red_{j,i} \vee lr_{j,i} \vee rr_{j,i} \vee \neg la_{j,i} \vee \neg ra_{j,i}) \\ &\Leftarrow (red_{j,i} \vee \neg edge_{j,i} \vee \neg lr_{j,i}) \wedge (red_{j,i} \vee \neg edge_{j,i} \vee \neg rr_{j,i}) \\ &\quad \wedge (red_{j,i} \vee \neg edge_{j,i} \vee \neg la_{j,i} \vee ra_{j,i}) \wedge (red_{j,i} \vee \neg edge_{j,i} \vee \neg ra_{j,i} \vee la_{j,i}) \end{aligned}$$

Now, in order to encode the red un-vanished edges at the moment of a contraction, to be able to restrict the maximum degree we introduce a new variable $reduv_{i,j,k}$. This is created at the moment node i is formed, and a red edge between the nodes j and k exists (and the nodes j and k of course still have not vanished). For each $i \in [n+1, 2n-d-2]$, for each $j \in [i]$, for each $k \in [j+1, i]$,

$$reduv_{i,j,k} \iff \neg vanish_{i,j} \wedge \neg vanish_{i,k} \wedge red_{j,k}$$

This is expressed in CNF as:

$$(\neg vanish_{i,j} \vee \neg reduv_{i,j,k}) \wedge (\neg vanish_{i,k} \vee \neg reduv_{i,j,k}) \wedge (red_{j,k} \vee \neg reduv_{i,j,k}) \\ \wedge (vanish_{i,j} \vee vanish_{i,k} \vee \neg red_{j,k} \vee reduv_{i,j,k})$$

Lastly we need to ensure that the maximum red degree at any point is at most d which is expressed as

$$\bigwedge_{\substack{i \in [n+1, 2n-d-2], \\ j \in [i]}} \left\langle \sum_{k \in [i], j \neq k} reduv_{i,min(j,k),max(j,k)} \leq d \right\rangle$$

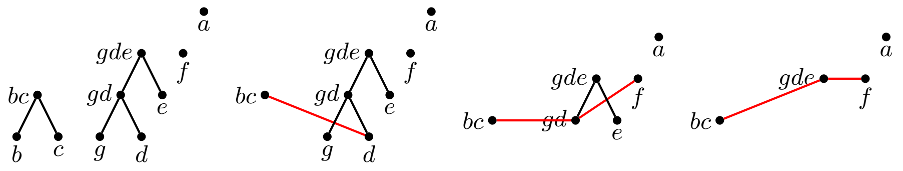

Fig. 2. The contraction forest and the sequence up to $2n - d - 2$ for $d = 2$ that occurs through the binary encoding for the 2-contraction sequence of Figure 1. Observe that for $d = 2$ we stop once 4 vertices remain since after the next contraction a graph on 3 vertices occurs which can have at most 2 red degree.

Theorem 1. *Given a graph G of order n and an integer d, we construct in polynomial time a CNF formula which is satisfiable if and only if $tww(G) \leq d$, and has $\mathcal{O}(n^3)$ clauses.*

Proof. We first prove that given an assignment of the variables that satisfies the SAT formula corresponding to a graph G, we are able to build a d-contraction sequence for G. First, we create the contraction tree following the parent-child relations occuring by the assignment values for the variables $lc_{i,j}$, $rc_{j,i}$. Then, we construct a contraction sequence based on the children of $n_1, \ldots, 2n-d-2$. By the construction of our formula we ensure that all edges of G are encoded. Also, for every contraction node i, the assignment denoting whether an edge between its children and other nodes exists, and the red edges created though the contraction of a pair of nodes, are counted, as the clauses equivalent to $edge_{j,i} \iff (la_{j,i} \vee ra_{j,i}) \wedge \neg vanish_{i,j}$ are satisfied. Lastly, if during this contraction sequence the red degree becomes greater than d, the clauses equivalent to $\bigwedge_{\substack{i\in[n+1,2n-d-2],\\ j\in[i]}} \sum_{k\in[i],j\neq k} reduv_{i,min(j,k),max(j,k)} \leq d$ are not satisfied, leading to a contradiction. Hence, the sequence occurring from contracting the child nodes of $n+1,\dots,2n-d-2$ is indeed a d-contraction sequence of G.

Given a d-contraction sequence of G we construct an assignment that satisfies the formula corresponding to G as follows: For the i^{th}-contraction pair, with the contracted vertices being x and y, where $i \leq 2n-d-2$ we assign 1 to $lc_{x,n+i}, rc_{y,n+i}$ and 0 for the other values of i. Hence, we have "built" our contraction tree. We also initialize $edge_{i,j}$ to 1 for the adjacent pairs of vertices in G, and $red_{i,j}$ to 0 for all pairs of vertices, to encode G. Similarly for right and left adjacencies of the nodes of T. We then follow the changes each contraction causes on G, and assign truth values to the respective variables. For example, for each variable $vanish_{p,i}$ if the vertex i has been contracted when it is p's turn to participate in some contraction, we assign 1 to it. Similarly, we assign 1 to $reduv_{i,j,k}$ for the red edges that survive each contraction. Notice that because a d-contraction sequence of G is given, due to the construction of our formula, the assignment created so far also satisfies the clauses of $\bigwedge_{\substack{i\in[n+1,2n-d-2],\\ j\in[i]}} \sum_{k\in[i],j\neq k} reduv_{i,min(j,k),max(j,k)} \leq d$, satisfying the CNF corresponding to G.

We now analyze the size of the CNF formula created from a graph G and an integer d, by counting the clauses occurring from each information encoding. For initializing $E(G)$, we use one variable for each edge, one for each non-edge and one to denote the absence of red edges, hence $\mathcal{O}(n^2)$ in total. To encode the contraction tree we use, for the "one-parent property", for each $i \in [2n-d-3]$ at least $2(n-d-2)$ variables, meaning $\mathcal{O}(n^2)$, and for the "binary property", for each $j \in [n+1, 2n-d-2]$, j variables for each child, hence $\mathcal{O}(n^2)$ in total. When converted to CNF (the same holds for bounding the red degree by d, by [13]) we get $\mathcal{O}(n^2)$ clauses. For $vanish_{p,i}$ we get two literals for each leaf, at most $2n-d-2$ for each of the internal nodes $2n-d-2$ for the "self variables", meaning $\mathcal{O}(n^2)$ in total. To encode right and left adjacencies (edges between the vertices represented by the children of a node and vertices represented by other nodes) we have $\mathcal{O}(n^2)$ variables to choose which child to refer to, and $\mathcal{O}(n)$, meaning $\mathcal{O}(n^3)$ in total.

When encoding the existence of an edge as a relation of right and left adjacencies and the variable $vanish$, we get 4 clauses for each edge variable, and 4 for the existence of each red edge, so $\mathcal{O}(n^2)$ clauses in total. Similarly 4 clauses are created for each variable $reduv$ that represents the existence of red edges at the moment of a contraction, making them $\mathcal{O}(n^3)$ in total. Lastly, for bounding the maximum red degree by d we need at most $\mathcal{O}(n^2)$ variables, which also produce $\mathcal{O}(n^2)$ clauses. Hence, the size of the formula produced by our encoding is $\mathcal{O}(n^3)$.

name	n	m	tww	binary	relative	name	n	m	tww	binary	relative
EX_001	19	64	6	12.86	**9.23**	ER_0.1	30	44	3	304.23	**37.23**
EX_002	20	69	6	4.27	**1.15**	ER_0.2	30	100	6	**100.81**	933.68
EX_003	25	97	6	**48.64**	85.84	ER_0.3	30	120	7	**587.99**	703.2
EX_004	25	181	7	107.33	**68.72**	ER_0.4	-	-	-	-	-
EX_006	28	131	7	1390.23	**480.01**	ER_0.5	30	236	9	375.13	**319.4**
EX_007	28	205	7	**178.24**	181.24	ER_0.6	-	-	-	-	-
EX_008	28	210	10	26.83	**8.68**	ER_0.7	30	293	8	**409.17**	-
EX_009	28	228	7	98.99	**68.77**	ER_0.8	30	342	6	**502.77**	-
EX_010	28	235	6	735.61	**438.79**	ER_0.9	30	381	4	-	**1418.76**
EX_011	29	174	8	-	**1077.78**	P_013	13	39	6	0.25	**0.1**
EX_012	29	180	8	-	**1163.65**	P_017	17	68	8	0.83	**0.25**
EX_013	30	155	8	-	**1498.19**	P_029	29	203	14	14.27	**7.52**
EX_014	30	175	8	1026.46	**873.93**	P_037	37	333	18	85.19	**35.83**
EX_015	30	178	8	**414.99**	858.01	P_041	41	410	20	175.01	**58.7**
EX_016	30	195	8	-	**1527.49**	P_053	53	689	26	660.1	**352.25**
EX_017	30	207	8	-	**409.64**						
EX_018	31	52	3	648.12	**175.37**						
EX_019	32	90	5	**1578.88**	-						
EX_031	48	80	3	**974.02**	1206.32						
EX_034	51	240	4	484	**16.66**						
EX_035	52	53	2	253.07	**135.8**						

Table 1. Results for the PACE dataset, random graphs, and Paley graphs. The numbers under "binary" and "relative" are the running times of the encodings, in seconds.

name	n	m	tww	time, absolute	time, binary	factor
tiny001.gr	10	9	1	1.48	**0.09**	x16
tiny002.gr	10	10	2	3.7	**0.13**	x29
tiny005.gr	25	40	3	>10 hours	**21.35**	>x2000
tiny007.gr	14	13	2	3131.77	**1.16**	x2689
tiny008.gr	10	15	4	65.80	**0.07**	x937
tiny009.gr	7	7	1	**0.01**	0.03	x0.33

Table 2. Comparison between binary encoding and *absolute* encoding on the tiny dataset of PACE.

4 Experiments

We implemented the binary SAT encoding presented above and run it on several datasets. For the implementation, we used Python 3.10.12 with PySAT 3.1.0, and Cadical as the specific SAT solver. The tests were run on a PC with AMD Ryzen 7 6800HS CPU, 16 GB RAM, and Ubuntu 22.04, using only a single thread. We also include an implementation of the relative encoding of [14], and compare the performance of our binary SAT encoding with the relative encoding. To make a fair comparison, both encodings were run using exactly the same settings and auxiliary code. We generally do not compare with the absolute encoding of [14] (except for Table 2), as it is hopelessly inefficient compared to both relative encoding and our binary encoding. We also implemented modular decomposition as the standard preprocessing step for computing twin-width; for detailed description of the preprocessing see [14]. The implementations and the testing data are provided in the supplementary material.

Next, we describe the datasets that were used for testing, and provide tables that compare performance of the binary encoding and the relative encoding. In all tests the computed twin-width value is the same for both encodings (as both are shown to output the optimal value of twin-width), so we only compare the time used by each encoding. In the results that we list, time is always measured in seconds. All tests were run with a time limit of 30 minutes (1800 seconds) and a memory limit of 10 gigabytes. Entries marked with "-" are used to denote that the solver exceeded time and/or memory limit on the corresponding instance.

Random graphs. We first construct Erdős–Rényi graphs $G(n, p)$ where $n = 30$, and p ranges from 0.1 to 0.9 win an increment of 0.1. The graph $G(n, p)$ contains n vertices, and each edge is created with probability p. The results for Erdős–Rényi graphs are listed in Table 1 in the "ER" rows. It is interesting to note that while the relative encoding performs slightly better in the uniform setting ($p = 0.5$), and considerably better when the graph is very sparse ($p = 0.1$) or very dense ($p = 0.9$), the binary encoding vastly outperforms the relative encoding in the intermediate cases ($p \in \{0.2, 0.3, 0.7, 0.8\}$).

Moreover, the performance of both solvers is worse for larger p compared to smaller p; this implies that neither encoding is exploiting to the fullest the property that the twin-width of a graph is always equal to the twin-width of its complement. That is, computing the twin-width of $G(n, p)$ should be equally hard as computing the twin-width of $G(n, 1 - p)$, since in the complement of $G(n, p)$ each edge exists independently with probability $1 - p$.

PACE 2023 challenge. We next compare the encodings on several instances from the PACE challenge. We first compare the binary encoding and the absolute encoding of [14] on the "tiny" sample set of the challenge[4]. The results are shown in Table 2, and highlight that the performance of the absolute encoding is much worse that that of the binary encoding, despite the same asymptotic size of the encoding. For this reason, we do not include absolute encoding in the other tests.

Table 1 then shows the result of comparison between the binary and the relative encoding on the regular instances of the challenge, see rows starting with "EX". While the original dataset contains 200 graphs[5], we only list the instances where at least one of the two solvers was able to compute the twin-width under the time and memory limit of 30 minutes and 10GB. While the relative encoding is generally faster on this dataset, the binary encoding is still able to perform considerably better on several instances. This further shows that the binary encoding is conceptually different from the relative, and may be used to deal with the instances where the relative encoding is not efficient.

[4] https://pacechallenge.org/2023/tiny-set.pdf

[5] All instances are available at https://pacechallenge.org/2023/.

Paley graphs. Finally, following the experiments of [14], we test the encodings on Paley graphs. We construct Paley graphs for several prime numbers; for a prime p with $p \equiv 1(\text{mod } 4)$, Paley graph is a $\frac{p-1}{2}$-regular graph on p vertices. The vertices of the Paley graph are associated with the elements of the unique finite field of order p, and the edge between x and y appears if and only if $x - y$ is a square in the field; in the case $p \equiv 1(\text{mod } 4)$, $x - y$ is a square if and only if $y - x$ is a square so the edges are symmetric. Last part of Table 1 lists the results. While the relative encoding is generally faster, it is interesting to observe that the memory usage of the binary encoding is lower on the Paley graphs. This may be attributed to the smaller size of the encoding, although this effect does not show on the other instances.

5 Conclusion

In this work, we introduced a novel SAT encoding for computing twin-width of a graph. Theoretically, we have shown that the encoding is sound, and that the size of the encoding is smaller than that of the state-of-the-art relative encoding of [14]. Further, we conducted experiments on several datasets, comparing the performance of the binary encoding, and the relative and absolute encoding of [14]. Similar to relative encoding, binary encoding vastly outperforms the absolute encoding. Comparing the binary and relative encoding, the results suggest that neither encoding dominates the other, while in many cases the binary encoding is much more efficient than the relative encoding (although the converse also happens). Therefore, the introduction of the binary encoding indeed improves the ability to compute twin-width in practice. This also motivates further work on comparing different twin-width encodings empirically, identifying families of instances where twin-width can be computed efficiently with either encoding.

References

1. Balabán, J., Hliněný, P.: Twin-Width Is Linear in the Poset Width. In: Golovach, P.A., Zehavi, M. (eds.) 16th International Symposium on Parameterized and Exact Computation (IPEC 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 214, pp. 6:1–6:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.IPEC.2021.6
2. Bergé, P., Bonnet, É., Déprés, H.: Deciding twin-width at most 4 is np-complete. In: Bojanczyk, M., Merelli, E., Woodruff, D.P. (eds.) 49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France. LIPIcs, vol. 229, pp. 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.ICALP.2022.18
3. Bonnet, E., Déprés, H.: Twin-width can be exponential in treewidth. J. Comb. Theory Ser. B **161**(C), 1–14 (may 2023). https://doi.org/10.1016/j.jctb.2023.01.003
4. Bonnet, É., Geniet, C., Kim, E.J., Thomassé, S., Watrigant, R.: Twin-width II: small classes. In: Marx, D. (ed.) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021. pp. 1977–1996. SIAM (2021). https://doi.org/10.1137/1.9781611976465.118

5. Bonnet, É., Geniet, C., Kim, E.J., Thomassé, S., Watrigant, R.: Twin-width III: max independent set, min dominating set, and coloring. In: Bansal, N., Merelli, E., Worrell, J. (eds.) 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference). LIPIcs, vol. 198, pp. 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.ICALP.2021.35
6. Bonnet, E., Kim, E.J., Reinald, A., Thomassé, S., Watrigant, R.: Twin-Width and Polynomial Kernels. In: Golovach, P.A., Zehavi, M. (eds.) 16th International Symposium on Parameterized and Exact Computation (IPEC 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 214, pp. 10:1–10:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.IPEC.2021.10
7. Bonnet, É., Kim, E.J., Thomassé, S., Watrigant, R.: Twin-width I: tractable FO model checking. J. ACM **69**(1), 3:1–3:46 (2022). https://doi.org/10.1145/3486655
8. Ganian, R., Pokrývka, F., Schidler, A., Simonov, K., Szeider, S.: Weighted model counting with twin-width. In: Meel, K.S., Strichman, O. (eds.) 25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel. LIPIcs, vol. 236, pp. 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.SAT.2022.15
9. Guillemot, S., Marx, D.: Finding small patterns in permutations in linear time. In: Chekuri, C. (ed.) Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014. pp. 82–101. SIAM (2014). https://doi.org/10.1137/1.9781611973402.7
10. Jacob, H., Pilipczuk, M.: Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. In: Bekos, M.A., Kaufmann, M. (eds.) Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13453, pp. 287–299. Springer (2022). https://doi.org/10.1007/978-3-031-15914-5_21
11. Král, D., Lamaison, A.: Planar graph with twin-width seven. CoRR **abs/2209.11537** (2022). https://doi.org/10.48550/arXiv.2209.11537
12. Král', D., Lamaison, A.: Planar graph with twin-width seven. European Journal of Combinatorics p. 103749 (2023). https://doi.org/https://doi.org/10.1016/j.ejc.2023.103749
13. Marques-Silva, J., Lynce, I.: Towards robust CNF encodings of cardinality constraints. In: Bessiere, C. (ed.) Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4741, pp. 483–497. Springer (2007). https://doi.org/10.1007/978-3-540-74970-7_35
14. Schidler, A., Szeider, S.: A SAT approach to twin-width. In: Phillips, C.A., Speckmann, B. (eds.) Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2022, Alexandria, VA, USA, January 9-10, 2022. pp. 67–77. SIAM (2022). https://doi.org/10.1137/1.9781611977042.6