Computing Twin-width with SAT and Branch & Bound

André Schidler , **Stefan Szeider**
Algorithms and Complexity Group, TU Wien, Vienna, Austria
{aschidler, sz}@ac.tuwien.ac.at

Abstract

The graph width-measure twin-width recently attracted great attention because of its solving power and generality. Many prominent NP-hard problems are tractable on graphs of bounded twin-width if a certificate for the twin-width bound is provided as an input. Bounded twin-width subsumes other prominent structural restrictions such as bounded treewidth and bounded rank-width. Computing such a certificate is NP-hard itself, already for twin-width 4, and the only known implemented algorithm for twin-width computation is based on a SAT encoding.

In this paper, we propose two new algorithmic approaches for computing twin-width that significantly improve the state of the art. Firstly, we develop a SAT encoding that is far more compact than the known encoding and consequently scales to larger graphs. Secondly, we propose a new Branch & Bound algorithm for twin-width that, on many graphs, is significantly faster than the SAT encoding. It utilizes a sophisticated caching system for partial solutions. Both algorithmic approaches are based on new conceptual insights into twin-width computation, including the reordering of contractions.

1 Introduction

Recently, Bonnet et al. [Bonnet *et al.*, 2020] discovered the fundamental graph width measure *twin-width* based on a novel graph decomposition based on sequences of vertex contractions. They showed that bounded twin-width generalizes many previously known graph classes for which important problems are tractable, including first-order model checking (an important meta-problem in computational logic). The notion of twin-width is also helpful for problems beyond NP, like bounded-ones weighted model counting [Ganian *et al.*, 2022]. Most notably, graphs of bounded treewidth, bounded clique-width, and planar graphs all have bounded twin-width [Bonnet *et al.*, 2020]; hence twin-width provides a common generalization of other diverse notions. Due to its appealing properties, twin-width has already become the topic of extensive research [Bonnet *et al.*, 2022f; Bonnet *et al.*, 2021a; Bonnet *et al.*, 2021b; Bonnet *et al.*, 2022c; Bonnet *et al.*, 2022c; Bonnet *et al.*, 2022d; Bonnet *et al.*, 2022e; Bonnet *et al.*, 2022a; Bonnet *et al.*, 2022b; Dreier *et al.*, 2022; Bergé *et al.*, 2022; Gajarský *et al.*, 2022; Pilipczuk *et al.*, 2022; Balabán *et al.*, 2022; Balabán and Hlinený, 2021; Jacob and Pilipczuk, 2022; Ahn *et al.*, 2022].

All the mentioned algorithmic results for graphs of bounded twin-width require a certificate that the input graph's twin-width is bounded. So far, twin-width has been mainly studied from a theoretical and asymptotic perspective and very little is known about the computational aspects of twin-width. An exception are the SAT encodings proposed by Schidler and Szeider [2022], which have been utilized for experiments on twin-width of model counting instances [Ganian *et al.*, 2022] but still only scale to relatively small instances. The SAT encoding provided experimental insights and gave rise to the conjecture that Paley graphs of order n have twin-width $(n-1)/2$, later verified by Ahn *et al.* [2022]. Because of the great interest in the practical computation of twin-width, the organizers of the 2023 PACE competition[1] have chosen twin-width as the target problem.

This paper aims to advance our knowledge about the computational aspects of twin-width. In particular, we ask (1) whether the SAT encodings can be improved to admit better scalability and (2) whether algorithmic approaches different from SAT encodings are feasible and effective. We also ask for (3) structural results that allow us to prune the search space for twin-width decompositions explored.

Indeed, we could make considerable progress in all three points and provide an empirical evaluation..

1. We propose a new SAT encoding that is significantly more compact than the known SAT encodings and allows faster solving times. The encoding's small size results in improved scalability and the encoding's ability to find upper bounds for larger graphs.
2. We propose a new Branch & Bound algorithm BB-CCH that incorporates several new methods proposed in this paper. The caching and re-using of previous computations is a key contributor to BB-CCH's outstanding performance. BB-CCH is considerably faster than the SAT

[1] Parameterized Algorithms and Computational Experiments Challenge (PACE), https://pacechallenge.org/2023/

encodings and can compute reasonable upper bounds for large graphs far out of reach for the SAT encodings.

3. We establish several results that can help to speed up the search for twin-width decompositions. The results include pre-and inprocessing techniques based on the new notion of a tri-twin and a result that allows us to assume a specific order of contractions (the elementary steps of twin-width decompositions) without losing optimality. Both help to significantly reduce the search space and speed up the twin-width computation.
4. We evaluate our algorithms on two classes of benchmark graphs, structured graphs and graphs from the treewidth lib benchmark suite. The results show that our new SAT encoding scales better than the previous encodings, particularly for structured graphs. Furthermore, BB-CCH solves more instances in a shorter amount of time than any of the SAT encodings.

2 Preliminaries

A *trigraph* is an undirected graph G with vertex set $V(G)$ whose edge set $E(G)$ is partitioned into a set $B(G)$ of *black edges* and a set $R(G)$ of *red edges*. We consider an ordinary graph a trigraph with all its edges black. W.l.o.g., we assume that $V(G)$ is lexicographically ordered. The set $N_G(v)$ of neighbors of a vertex v in a trigraph G consists of all the vertices adjacent to v by a black or red edge. We call $u \in N_G(v)$ a *black neighbor* of v if $uv \in B(G)$, we call it a *red neighbor* if $uv \in R(G)$, and denote the set of black and red neighborhoods of v by $N_{G,B}(v)$ and $N_{G,R}(v)$ respectively. Whenever G is clear from context, we drop G from the subscript. The *red degree* of a vertex $v \in V(G)$ of a trigraph G is $|N_{G,R}(v)|$. A d-trigraph is a trigraph where each vertex has red degree at most d, denoted by $r(G) = d$.

A bijection $\pi{:}V(G) \to V(H)$ between the vertex sets of trigraphs G and H is an *isomorphism* if for all $u, v \in V(G)$ it holds that $uv \in X(G)$ iff $\pi(u)\pi(v) \in X(G)$, for $X \in \{B, R\}$. If an isomorphism between G and H exists, the trigraphs are *isomorphic*. If $G = H$ then π is an *automorphism*. $V(G)$ is partitioned into *orbits* where $u, v \in V(G)$ belong to the same orbit if $\pi(u) = v$ for some automorphism π of G.

A trigraph G' is *derived* from a trigraph G by a *contraction* $u \to v$, where u is being merged into v, as follows: $V(G') = V(G) \setminus \{u\}$ and every vertex in the *symmetric difference* $N_G(u) \triangle N_G(v)$ is made a red neighbor of v. If a vertex $x \in N_G(u) \cap N_G(v)$ is a black neighbor of both u and v, then v is made a black neighbor of x; otherwise, v is made a red neighbor of x. The other edges (not incident with u or v) remain unchanged. We denote a trigraph obtained from trigraph G by contracting u and v as $G^{u \to v}$, and in the contraction $u \to v$, u is the *contracted* vertex, and v is the *contraction* vertex. $G^{u \to v}$ is isomorphic to $G^{v \to u}$, as only the name of the contraction vertex changes.

Twin-Width via d-Elimination Sequences. We give here a definition of twin-width via *d-elimination sequences*, slightly modifying the definition given by Schidler and Szeider [2022], as this definition is better suited for our purposes.

Let G be a trigraph, T a rooted forest—every tree in T is a rooted tree—with $V(T) \subseteq V(G)$, set of non-roots $N(T)$, and $\prec$ a linear ordering of $N(T)$, where $u \prec v$ holds for any two vertices $u, v \in N(T)$ such that v is the parent of u in T. We call T a *contraction forest*, $\prec$ an *elimination ordering*, and the pair $D = (T, \prec)$ a *twin-width decomposition* of G. Let $V' = V(G) \setminus N(T)$, $k = |N(T)|$, when we write $V(G) = \{v_{1,D}, \ldots, v_{k,D}\} \cup V'$ such that $v_{1,D} \prec \cdots \prec v_{k,D}$, then T and G define a sequence of trigraphs $G_0, \ldots, G_k$ with $V(G_i) = \{v_{i+1,D}, \ldots, v_{k,D}\} \cup V'$. We denote by $p_{i,D}$ the parent of $v_{i,D}$ in T. We define G_i recursively as follows. For $i = 0$, we set $G_0 = G$, and for $1 \leq i \leq k$ we set $G_i = G_{i-1}^{v_i \to p_{i,D}}$.

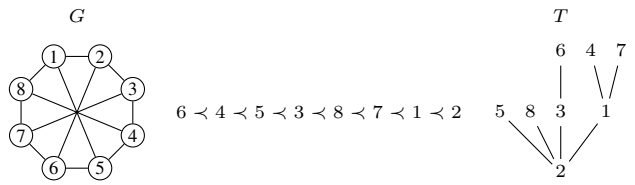

Figure 1: A graph G with eight vertices and a twin-width decomposition $(T, \prec)$ of width 2. Figure 2 shows the 2-elimination sequence induced by $(T, \prec)$.

We call the sequence $G = G_0, \ldots, G_k$ the *elimination sequence* for G defined by the twin-width decomposition $(T, \prec)$; if for an integer d, all the G_i have a maximum red degree $\leq d$, we call $G = G_0, \ldots, G_k$ a d-elimination sequence. The *width* of the twin-width decomposition $(T, \prec)$ of G is the smallest integer d such that $(T, \prec)$ defines a d-elimination sequence.

Whenever $k = |V(G)| - 1$, we call $(T, \prec)$ a *full twin-width decomposition*, and consequently $\prec$ a *full elimination ordering*, $G_0, \ldots, G_k$ a *full elimination sequence*, and T a *full contraction forest*. Observe that, in this case, T is a tree rooted at a single vertex. The twin-width of a trigraph G, denoted *tww*(G), is the minimum width over all See Figures 1 and 2 for examples. Recognizing graphs of twin-width 4 is NP-complete [Bergé *et al.*, 2022].

Partitions. The contractions of a graph G induce a *partition* on $V(G)$ as follows. We view every vertex $v \in V(G)$ as labeled by the singleton $L_G(v) = \{v\}$. After contracting two vertices $u \to v$, we label the contraction vertex v as $L_{G^{u \to v}}(v) = L_G(u) \cup L_G(v)$, and all other labels remain the same. Hence, the labels of a trigraph $G = G_0, \ldots, G_k$ derived from a graph G are always a partition of $V(G)$. For $0 \leq i \leq k$ we refer to the partition of G_i as $P(G_i) = \{ L_{G_i}(v) \mid v \in V(G_i) \}$.

3 Theory

For a graph with n vertices exist $\frac{n! \cdot (n-1)!}{2^n}$ many possible sequences of contractions, each of them corresponding to a full twin-width decomposition. In this section, we discuss how we can reduce the number of twin-width decompositions we have to consider and speed up the twin-width computation for a given trigraph. We will use these results for our algorithms SAT-CPT and BB-CCH.

We first state two facts, let G be a trigraph, $u, v \in V(G)$.

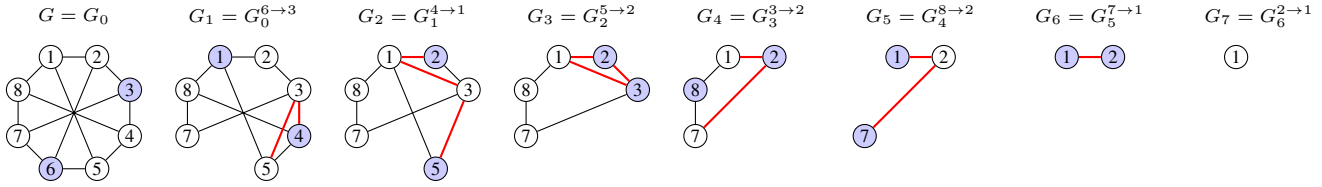

Figure 2: 2-elimination sequence induced by the twin-width decomposition from Figure 1.

Fact 3.1. *The red degree of any vertex $w \in V(G) \setminus \{u, v\}$ can increase at most by one in $G^{u\to v}$.*

Fact 3.2. *The red degree of any vertex $w \in V(G) \setminus \{u\}$ can decrease at most by one in $G^{u\to v}$.*

3.1 Isomorphic Trigraphs

Trigraph isomorphisms can be used to eliminate many possible twin-width decompositions from consideration. Whenever we have a twin-width decomposition $D = (T, \prec)$ inducing a d-elimination sequence $G = G_0, \dots, G_k$ and we have a full d-elimination sequence $G = G'_0, \dots, G'_{n-1}$, such that G_k and G'_k are isomorphic, then, $G = G_0, \dots, G_k, G'_{k+1}, G'_{n-1}$ is a full d-elimination sequence. Hence, we complete T and $\prec$ using the full twin-width decomposition, and we do not have to go through all possible contractions to complete D. Furthermore, whenever we cannot complete D to a full twin-width decomposition of width d, we know that no d-elimination sequence $G = G'_0, \dots, G'_j$, where, for any $i \in \{1, \dots, j\}$, G'_i is isomorphic to G_i, can be extended to a full d-elimination sequence.

Since checking if trigraphs are isomorphic can be computationally expensive, We give an easy sufficient criterion with the following theorem:

Theorem 1. *Given twin-width decompositions $D = (T, \prec)$ and $D' = (T', \prec')$ inducing the d-elimination sequences $G = G_0, \dots, G_k$ and $G'_0, \dots, G'_k$ respectively. Whenever $P(G_k) = P(G'_k)$, then G_k and G'_k are isomorphic.*

The theorem follows from the definition of *partition coarsening* introduced by Bonnet *et al.* [2020]. Hence, whenever two trigraphs represent the same partition, they are isomorphic. The opposite is not necessarily true, different partitions may represent the same trigraph. The following observations lead to complementary isomorphism checks:

Observation 3.1. *Consider a trigraph G, vertices $u, v, w \in V(G)$, and let $A = R(G^{u\to w}) \triangle R(G)$ and $B = R(G^{v\to w}) \triangle R(G)$. If either $A = \{vw\}$ and $B = \{uw\}$ or $A = B = \emptyset$, the trigraphs $G^{u\to w}$ and $G^{v\to w}$ are isomorphic.*

Observation 3.2. *If two vertices $u, v \in V(G)$ belong to the same orbit, then for each $u' \in V(G)$ there exists a $v' \in V(G)$ such that $G^{u\to u'}$ and $G^{v\to v'}$ are isomorphic.*

3.2 Twin Pre- and In-processing

We propose a new pre- and inprocessing method and extend the definition of twins from graphs to trigraphs.

Definition 1. *Two different vertices $u, v \in V(G)$ of a trigraph G are* tri-twins *if the following two properties hold: (i) $(N_B(u) \triangle N_B(v)) \setminus \{u, v\} = \emptyset$, and (ii) $N_R(u) \setminus \{v\} \subseteq N_R(v)$ or $N_R(v) \setminus \{u\} \subseteq N_R(u)$.*

Observation 3.3. *Contracting tri-twins u, v does not increase the red degree of any vertex, and the red degree of the contraction vertex is at most the red degree of u or v.*

Tri-twins have another interesting property, as the following theorem shows.

Theorem 2. *Let G_j be a trigraph derived from some graph G via contractions, and u, v be tri-twins in G_j. Then, $tww(G_j) \geq tww(G_j^{u\to v})$.*

The proof proceeds by showing that a full d-elimination sequence $G = G_0, \dots, G_j, \dots G_{n-1}$ can be modified such that u and v are contracted at position $j + 1$, which yields a d-elimination sequence $G_j^{u\to v} = G'_{j+1}, \dots, G'_{n-1}$. Because contraction and contracted vertex are interchangeable, we can assume that $V(G_{n-1}) = V(G'_{n-1}) = \{v\}$.

Theorem 2 allows us to apply the following pre- and inprocessing: whenever we have a partial elimination sequence up to G_j that we are trying to extend to G_{j+1} and find tri-twins in G_j, we can immediately contract them and disregard all other contractions. This can considerably reduce the number of elimination sequences we need to explore.

3.3 Ordering of Contractions

In general, the ordering of the contractions in an elimination sequence matters. Therefore, given a twin-decomposition $(T, \prec)$ of width d, changing $\prec$ may cause the width to increase. While generally true, the ordering of the contractions often does not matter, and reordering of many contractions still maintains the same width. This is important, as only having to consider different sets of contractions is less complex than also considering their ordering. Therefore, criteria that help us distinguish between cases where the ordering matters and does not matter, can significantly reduce the search space.

We give some intuition, before formalizing our criterion. Consider a twin-decomposition $D = (T, \prec)$ inducing a d-elimination sequence $G = G_0, \dots, G_{k-1}$. The key to this idea is Theorem 1: the maximum red degree of any vertex in G_{k-1} only depends on the partition induced by the contractions. In particular, the order of the contractions does not matter, as reordering the contractions will still induce the same partition.

We can view the red degree of a vertex $v \in V(G_{k-1})$ by looking at $\Sigma_{j=1}^{k-1} |N_{G_j,R}(v)| - |N_{G_{j-1},R}(v)|$, i.e., the change of red degree from contraction to contraction. This allows us to incrementally maintain the red degrees: when adding one summand after the other, the intermediate result never exceeds d. Further, as long as this invariant holds for all vertices, we can freely move around the contractions.

In our criterion, we consider the contraction $u \to v$ to extend our elimination sequence $G = G_0, \ldots, G_{k-1}$ to $G_k = G_{k-1}^{u \to v}$. Let $d' = \max\{r(G_k), r(G_k^{u \to v})\}$. The question we want to answer is: are there any $v_{i,D} \in (V(G) \setminus V(G_k)) \setminus \{u\}$ such that there exists a twin-width decomposition $D' = (T', \prec')$ of G_{i-1} that has width d' and $v_{i,D'} = u$?

We give a general condition in the following theorem, but first, we introduce some necessary notations. Let $D = (T, \prec)$ be a twin-width decomposition that induces a d-elimination sequence $G = G_0, \ldots, G_k$. For all $1 \leq i \leq k$ we define

$$R_i = (N_{G_i,B}(v_{k,D}) \triangle N_{G_i,B}(p_{k,D})) \setminus \{v_{k,D}, p_{k,D}\},$$
$$N_i = (N_{G_i,R}(v_{k,D}) \cup N_{G_i,R}(p_{k,D})) \setminus \{v_{k,D}, p_{k,D}\}.$$

R_i is the set of new red neighbors created when contracting $v_{k,D}$ and $p_{k,D}$ in G_i. N_i is the set of existing red neighbors of the two vertices. We can now formulate our theorem.

Theorem 3. *Let $G = G_0, \ldots, G_k$ be the d-elimination sequence induced by a twin-width decomposition $D = (T, \prec)$ for trigraph G, and let $j_0 \in \{0, \ldots, k-1\}$ be the largest index such that at least one of the following conditions holds:*

1. *For some $w \in (R_{j_0} \setminus N_{j_0}) \cup \{v_{j_0}\}$ it holds that:*
   (a) *$|N_{G_{j_0-1},R}(w)| = d$, and*
   (b) *$w = v_{j_0}$ or $|N_{G_{j_0},R}(w)| = d - 1$,*
2. *$|R_{j_0} \cup N_{j_0}| > d$,*
3. *$j_0 = 0$.*

Then, for all $j_0 < j < k$, there exists a twin-width decomposition D_j of G of width d where $v_{j,D_j} = v_{k,D}$ and $p_{i,D_j} = p_{k,D}$.

Proof. We define for all $1 \leq j \leq k$ a twin-width decomposition $D_j = (T_j, \prec_j)$ as follows:

$$v_{i,D_j} = \begin{cases} v_{k,D} & i = j, \\ v_{i,D} & i < j, \\ v_{i-1,D} & i > j \end{cases} \quad p_{i,D_j} = \begin{cases} p_{k,D} & i = j, \\ p_{i,D} & i < j, \\ p_{k,D} & p_{i-1,D} = v_{k,D} \\ p_{i-1,D} & \text{otherwise.} \end{cases}$$

It can be verified that D_j is a twin-width decomposition of G of width d. □

Intuitively, we need to ensure that the reordering does not lead to any vertices red degree exceeding d. Condition 1 states that any vertex whose red degree is increased by $v_{k,D} \to p_{k,D}$ cannot have red degree d or is eliminated. Condition 2 ensures that $v_{k,D} \to p_{k,D}$ does not cause the red degree of $p_{k,D}$ to exceed d. The following corollary is a special case.

Corollary 1. *Let $D = (T, \prec)$ be a twin-width decomposition inducing a d-elimination sequence $G = G_0, \ldots, G_k$. Whenever $G_{k-2}^{v_{k,D} \to p_{k,D}}$ is a d-trigraph and for all vertices $v \in V(G_k)$ holds that either (i) $|N_{G_k,R}(v)| < d$, or (ii) $|N_{G_{k-1},R}(v)| = d$, or (iii) $|N_{G_{k-2},R}(v)| \leq |N_{G_{k-1},R}(v)|$, then D_{k-1} is a d-elimination sequence.*

This already gives us a strong criterion for finding equivalent reorderings of the given contractions. While it is easy to state a canonical criterion, such as that the contractions should occur in lexicographical order, except if it is necessary to execute them in a specific order. This can then be rather easily verified using Theorem 3. Converting a given elimination ordering into such a canonical ordering, on the other hand, is difficult, as each change in ordering requires re-evaluating the effects and which other vertices can then be reordered.

4 SAT Encoding

In this section, we propose our new compact SAT encoding SAT-CPT for determining the twin-width of a given graph G with $n = |V(G)|$ many vertices. Schidler and Szeider [2022] proposed two SAT encodings, a *relative* (SAT-SSZ) and an *absolute* encoding. While their absolute encoding was comparatively succinct, using only $O(n^3)$ many clauses, it cannot solve any non-trivial instances. The relative encoding uses $O(n^4)$ many clauses, which makes it considerably larger, but outperforms the absolute encoding. Our encoding SAT-CPT implements the basic idea of the absolute encoding in a way that is even more succinct and performs much better. Our encoding uses $O(n^3)$ many clauses, which makes it more succinct than the relative encoding, and the different cardinality encoding, discussed in Section 4, makes SAT-CPT also significantly more succinct than the absolute encoding.

In the remainder of this section, we present SAT-CPT by, given a graph G and an integer d, constructing a propositional formula $F(G, d)$ that is satisfiable if and only if $tww(G) \leq d$. For this purpose, we encode a twin-width decomposition $D = (T, \prec)$ and ensure that the induced elimination sequence $G = G_0, \ldots, G_{n-1}$ is a d-elimination sequence. We start with our representation of D, followed by our representation of $G_0, \ldots, G_{n-1}$, and finish the encoding with our handling of cardinality constraints. In the last part, we discuss optional techniques to speed up the solving process.

Encoding the Twin-Width Decomposition. We represent D by using for each $u \in V(G)$ the variables $v_{u,t}$ and $p_{u,t}$, where $v_{u,t}$ is true if and only if $v_{t,D} = u$ and $p_{u,t}$ is true if and only if $p_{t,D} = u$. Since the maximum red degree of a trigraph with k vertices is $k-1$, we can limit t to $1 \leq t < n-d-1$ and eliminate the last $d+2$ vertices in any order. Therefore, encoding D takes asymptotically $O(n^2)$ many variables, where the actual number depends on d.

Both sets of variables $p_{v,t}$ and $v_{v,t}$, $v \in V(G)$, $1 \leq t < n - d - 1$, are constrained such that exactly one vertex is eliminated at once and that $v_{t,D} \prec p_{t,D}$ holds.

This way of encoding D splits the definition of T between the variables $v_{u,t}$, and $p_{u,t}$, as $p_{t,D}$ can only be determined using both sets of variables. This is a significant difference to both the absolute and relative encoding by Schidler and Szeider [2022]: both encodings use variables that directly represent $p_{u,t}$. Our way of encoding T allows us to use only cubically many clauses, as we will discuss next.

Red Edges. Before we can constrain the red degrees of the vertices, we have to encode $G_1, \ldots, G_{n-d-2}$. We use variables $r_{t,u,w}$ for all $1 \leq t < n - d - 1$ and $u, w \in V(G)$ such that u is lexicographically smaller than w. A variable $r_{t,u,w}$ is true if and only if $uw \in R(G_t)$.

The asymptotic succinctness of SAT-CPT is due to how the semantics of $r_{t,u,v}$ is encoded. The red edges of a trigraph G_t with $1 \leq t < n-d-1$ are encoded in three steps. First, we maintain all red edges from G_{t-1}, which is straightforward. Next, we encode the new red edges between $p_{t,D}$ and vertices $u \in ((N_{G,B}(v_{t,D}) \triangle N_{G,B}(p_{t,D})) \cap V(G_t)) \setminus \{p_{t,D}\}$. The constraints are straightforward, as we can easily compute $N_{G,B}(v_{t,D}) \triangle N_{G,B}(p_{t,D})$ when creating the encoding, and add the condition that u must not yet have been eliminated.

Finally, we encode that for all $u \in N_{G_{t-1},R}(v_{t,D})$ it must hold that $u = p_{t,D}$ or $u \in N_{G_t,R}(p_{t,D})$, i.e., red neighbors get transferred from the contracted to the contraction vertex. Here, splitting the encoding of the contraction and the contracted vertex pays off. We use auxiliary variables $m_{t,u}$ for all $1 < t < n-d-1$ and $u \in V(G)$ that are true if and only if $uv_{t,D} \in R(G_{t-1})$. The desired property can then be easily encoded while maintaining the number of clauses in $O(n^3)$; here, SAT-SSZ requires $O(n^4)$ many clauses for encoding the elimination sequence.

Cardinality Constraints. We ensure that the red degree of any vertex never exceeds d using *cardinality constraints* that encode conditions of the form $|N_{G_t,R}(u)| \leq d$.

SAT-SSZ uses one cardinality constraint for each vertex in each trigraph of the elimination sequence. Hence, the encoding uses n^2 many cardinality constraints. Since a cardinality constraint on its own adds $O(n \cdot \log n)$ many variables and $O(n^2)$ many clauses [Bailleux and Boufkhad, 2003; Sinz, 2005], this adds $O(n^4)$ many clauses in total.

We use Facts 3.1 and 3.2 for a much more succinct solution. Since only the red degree of the contraction vertex can increase by more than one in a single contraction, we use the following trick. First, we constrain the red degree of the contraction vertex in each trigraph using *totalizer* cardinality constraints [Bailleux and Boufkhad, 2003]. This adds $n-d-2$ many cardinality constraints.

Next, we constrain each vertex using an *adapted sequence counter* [Sinz, 2005]. This sequence counter uses the fact that all vertices other than the contraction vertex can change their red degree by at most one during a single contraction. Hence, for each $v \in V(G)$ and $1 \leq t < n-d-1$, the counter changes as follows from $t-1$ to t: (i) whenever $v = p_{t,D}$, the counter uses the corresponding totalizer constraint to determine the red degree, (ii) whenever v has a red edge to both the contraction and contracted vertex in G_{t-1}, the counter decreases by one, (iii) whenever v has a red edge to either the contraction or contracted vertex in G_{t-1}, the counter stays the same, and (iv) whenever v has neither a red edge to the contraction nor the contracted vertex in G_{t-1}, but a red edge to the contraction vertex in G_t, the counter increases by one.

In total, we use only $2 \cdot |V| - d - 2$ many cardinality constraints, each adding $O(n^2)$ many clauses, which are significantly fewer variables and clauses compared to SAT-SSZ.

Symmetry Breaking. We use two ideas from Section 3 for symmetry breaking, often speeding up the solving process. First, we use Observation 3.2 to restrict the choices for $v_{u,1}$ to a single vertex per orbit. Further, we encode Corollary 1 for more symmetry breaking.

For encoding Corollary 1, we require the definition of a *critical vertex*: let $t \in \{2, \ldots, n-d-2\}$ and $u \in V(G)$, then u is a critical vertex in G_t, if $|N_{G_t,R}(u)| = d$, $|N_{G_{t-1},R}(u)| = d-1$, and $|N_{G_{t-2},R}(u)| = d$. We use this for vertices $u, w \in V(G)$ where u is lexicographically smaller than w. Whenever $v_{t,D} = u$ and $v_{t-1,D} = w$ it must also hold that either $|N_{G_t,R}(p_{t,D})| = d$ and $|N_{G_{t-1},R}(p_{t,D})| < d$, or any vertex $u \in V(G_t)$ is critical. This enforces a lexicographical ordering and thereby eliminates equivalent twin-width decompositions. We can express this with $O(n^2)$ many variables $c_{u,t}$, $2 \leq t < n-d-1$ and $u \in V(G)$, where $c_{u,t}$ is true if and only if u is critical in G_t.

It seems that encoding Theorem 3 would improve the performance even further. Unfortunately, this doubles the encoding size, while the performance gains do not compensate for the increase in size.

5 Branch & Bound

In this section, we propose our branch & bound algorithm BB-CCH, shown in Algorithm 1. The algorithm incrementally constructs a twin-width decomposition, adding one contraction in each recursive call and thereby constructing the elimination sequence. In each recursive call, whenever we find tri-twins, we immediately contract them (Theorem 2), and skip all other contractions (Line 5). Otherwise, the algorithm considers all possible contractions, and prioritizes those contractions that minimize the maximum red degree.

Central to the algorithm is the contraction filtering in Line 9. We can remove any contraction from consideration, where a rule applies. The given order ensures that computationally more expensive rules are applied later.

The simplest rule is that no trigraph should reach or exceed the upper bound, which is verified first. The *orbit* rule states that only the contractions of one vertex per orbit are considered; as stated in Observation 3.2. The *quick isomorphism* rule states that, given two contractions $u \to v$ and $u' \to v'$ with $u \neq u'$ that derive isomorphic trigraphs according to Observation 3.1, we only perform one and filter the other. Finally, we reuse previous results from a *cache*.

Caching for Isomorphisms. The algorithm's core idea is based on the idea to cache the results for partial elimination sequences. Whenever we have computed all possible extensions of G_i to a full elimination sequence, we know the lowest integer $best$ such that there exists a $best$-elimination sequence $G_i, \ldots, G_{n-1}$. In case, every possible extension reaches or exceeds the upper bound ub, we use $best = ub$. We then store $(G_i, best)$ in our cache, where G_i is the key used to access the result (Line 20). In later iterations, whenever the current trigraph is isomorphic to a trigraph in a cache entry, we can use the cached result instead of computing all extensions (Line 9).

Instead of the trigraph itself, we use the partition of the trigraph as the key due to Theorem 1. This requires less memory and comparing two partitions is faster than comparing two trigraphs. The potential downside of using the partition is that a trigraph can be represented by multiple partitions. Consequently, using the trigraph as the key would increase the chance for a successful lookup. In our experiments, using the trigraph over the partition as the key could increase the num-

Algorithm 1 Branch & Bound Algorithm for Twin-Width

Input: A graph G
Output: The twin-width of the graph.

Let $ub = |V(G)| - 1$
return TWW_RECURSION(G, 0)
function TWW_RECURSION(G_i, d)
  if $|V(G_i)| = 1$ **then return** 0
  if any $u, v \in V(G_i)$ are tri-twins (Theorem 2) **then**
    return TWW_RECURSION($G_i^{u \to v}$, d)
  $d \leftarrow \max(d, r(G_i))$
  ▷ Add all possible contractions to queue.
  $Q \leftarrow \{ (u, v) \mid u, v \in V(G_i), u \text{ lex smaller } v \}$
  ▷ Filter queue entries.
  for all $(u, v) \in Q$ where
    Maximum red degree of $G_i^{u \to v}$ reaches ub, or
    ($i = 0$ and automorphism (Observation 3.2)), or
    Quick isomorphism check (Observation 3.1), or
    $G_i^{u \to v}$ is in partition cache (Theorem 1), or
    Reordering succeeds (Theorem 3), or
    $G_i^{u \to v}$ is in graph isomorphism cache
    do
    $Q \leftarrow Q \setminus (u, v)$
  ▷ Process queue and recurse.
  $best \leftarrow ub$
  while $|Q| > 0$ **do**
    Select $(u, v) \in Q$ minimizing $r(G_i^{u \to v})$
    $Q \leftarrow Q \setminus (u, v)$
    $r \leftarrow$TWW_RECURSION($G^{u \to v}$, d)
    if $r < best$ **then**
      Clear caches.
      $best \leftarrow r$
      $ub \leftarrow \max(best, d)$
  Store $best$ in the two caches.
  return $\max(d, best)$

ber of successful cache lookups by less than 1%, which does not compensate for the decreased performance. We name this cache the *partition cache*.

We sometimes use a second cache, the *graph isomorphism cache*, where we use the *canonical* trigraph as the key: if two trigraphs are isomorphic than they have the same canonical trigraph. Hence, using the canonical trigraph increases the chances for a successful cache lookup. The canonical trigraph is found using the graph automorphism tool *nauty* [McKay and Piperno, 2014]. Unfortunately, computing the canonical trigraph takes very long compared to all other operations in a recursive call, and storing the whole trigraph takes up considerable memory. Therefore, we use it only for instance specific parts of the elimination sequence, where successful lookups are most likely.

Contraction Reordering. Let $G = G_0, \ldots, G_k$ be the current d-elimination sequence in our recursive call and we consider extending it to $G_{k+1} = G_k^{u \to v}$. Before we perform the contraction, we compute j_0 from Theorem 3. Hence, we check if there is a twin-width decomposition, where $u \prec v_{k,D}$ and the same contractions are performed, just in a different order, such that the twin-width decomposition still induces a d-elimination sequence.

We perform the following partition cache lookups, whenever $j_0 < k - 1$. For all $j_0 < j \leq k$, we check whether $P(G_j^{u \to v})$ is in our partition cache; if so, and the cached result is larger than d, we know that we cannot extend our current elimination sequence to a full d-elimination sequence.

This concludes the description of BB-CCH. Next, we discuss our experimental evaluation of our new methods.

6 Experiments

We implemented SAT-CPT and BB-CCH in C++ compiled with GCC 7.5.0[2]. We used nauty 2.8.6 [McKay and Piperno, 2014][3] as a libary for detecting trigraph isomorphisms and automorphisms, as well as Cadical [Biere *et al.*, 2020][4] as a SAT solver. We use standard hash maps as caches and a simple least recently used rule to manage a cache's memory consumption. We compare our methods to the relative SAT encoding (SAT-SSZ) by Schidler and Szeider [2022][5]. We ran the experiments on servers with two AMD EPYC 7402 CPUs, each with 24 cores running at 2.8 GHz, and using Ubuntu 18.04.

Structured Graphs. We start our experiments by looking at graph families where the number of vertices can be adapted. We choose from each family a graph whose size makes determining the twin-width difficult but possible. We examine the results for 5 different graphs: The *Paley-73*, *Grid* 8×6, and *Rook* 6×6 graphs as they were used by Schidler and Szeider [2022], as well as the *line graph* for the complete graph K_{10}, and the *Queen* 7×7 graph. Table 1 shows that SAT-CPT is indeed much more succinct than the previous encoding. This succinctness reflects itself in the runtime, where SAT-SSZ consistently required much more time than SAT-CPT. In Table 1, we omit the data for BB-CCH, as it solved all five instances within a second. Indeed, BB-CCH could solve the Rook 20×20, Queen 13×13, Line K_{25}, and Paley-421 graphs within a minute. Unfortunately, we could not determine the twin-width of larger grid graphs.

Treewidth Lib. We expand our comparison to the Treewidth Lib[6] instances. These instances are used as benchmarks for computing treewidth, a related width measure, and give a wider variety of graphs. We used all instances with 20 to 200 vertices and tested them using a 6-hour time limit and 32 GB memory limit. All three methods were most effective for graphs of up to 50 vertices. Out of these 25 instances, only one remained unsolved. Many of the remaining 217 instances were too difficult for any method. Nonetheless, the largest solved instance had 197 vertices.

[2]The source code is available under https://doi.org/10.5281/zenodo.7944399 and https://github.com/ASchidler/tww-bb. Experimental results are available under https://doi.org/10.5281/zenodo.7944391

[3]https://pallini.di.uniroma1.it/

[4]https://github.com/arminbiere/cadical

[5]https://github.com/ASchidler/twin_width

[6]http://www.cs.uu.nl/research/projects/treewidthlib/

Instance	*tww*	SAT-SSZ / SAT-CPT Vars	Clauses	t [s]
Grid 8×6	3	396 K/ 79 K	3.4 M/1.3 M	33655/1200
Paley-73	36	2.4 M/0.2 M	26 M/5.6 M	42910/ 789
Rook 6×6	10	209 K/ 35 K	1.4 M/0.6 M	855/ 9
Line K_{10}	14	479 K/ 65 K	3.0 M/1.3 M	29071/ 374
Queen 7×7	16	631 K/ 83 K	4.6 M/1.7 M	19947/9842

Table 1: Comparison of the SAT encodings on difficult structured graphs. M denotes million and K thousand.

Table 2 shows how many instances the different methods solved, how often an upper bound could be found, and how comparatively good this upper bound was. BB-CCH solved by far the most instances and was also much faster than the other methods: on all instances solved by both BB-CCH and SAT-SSZ (on average in 802 seconds), BB-CCH was faster (on average 107 in seconds). Interestingly, BB-CCH also produced upper bounds for all instances that were as good, and mostly better, than those obtained by the SAT encodings.

The comparison between the two SAT encodings suggests that SAT-SSZ performed better than SAT-CPT, as the former solved more instances. On closer look, SAT-CPT found more and better twin-width decompositions than SAT-SSZ. Furthermore, SAT-CPT also found more optimal twin-width decompositions but failed to show optimality. Hence, SAT-CPT is the better encoding for obtaining good twin-width decompositions, particularly for structured graphs. SAT-SSZ performed better for finding provably optimal twin-width decompositions on unstructured graphs. Finally, the scalability of SAT-SSZ is restricted by its size, while SAT-CPT can be used for larger graphs.

Difficult Instances. In the last part, we examine instances that were difficult for BB-CCH and how well the different methods proposed in Section 3 performed in practice. Table 3 shows the results. By far, most contractions are filtered due to reaching the upper bound (Exceeded). Interestingly, the ratio between recursive calls and successful partition cache lookups (PCache) is roughly equal for all instances, while the other methods performed very instance specific. This shows how effective caching is and that each method is important to adapt to different instances, as each method performed well for at least one instance in Table 3.

Method	Solved	Unique	Found	Better	Optimal
BB-CCH	44	26	242	204	48
SAT-CPT	12	0	151	63	29
SAT-SSZ	20	4	129	19	28

Table 2: Results for Treewidth Lib instances. *Solved* shows the number of optimal solutions and *Unique* how many could only be found by this method. *Found* shows the number of instances for which an upper bound could be found, and *Optimal* how often this upper bound happened to be the optimal solution. For BB-CCH and SAT-CPT, *Better* shows the number of times the upper bound was better than SAT-SSZ, and for SAT-SSZ, *Better* shows how often the upper bound was better than the one found by BB-CCH or SAT-CPT.

7 Conclusion

We have advanced our knowledge about the computational aspects of twin-width in several ways. We proposed two approaches for computing the twin-width of graphs, SAT-CPT and BB-CCH, utilizing new theoretical results on twin-width decompositions. These theoretical results are of independent interest as they might be helpful for other twin-width algorithms developed in the future.

SAT-CPT's advantage over its predecessor SAT-SSZ lies in being significantly more compact and scalable to larger instances. BB-CCH's advantages include being significantly faster than the SAT approaches and being able to compute good upper bounds for the twin-width of graphs much larger than SAT encodings can handle.

We hope that our results bring the compelling theoretical concept of twin-width a step closer to some practical use in the future. Further, as practical solvers can often help to verify theoretical results, we hope that computing the twin-width or good upper bounds for twin-width can provide new insights for researchers working on twin-width.

Future work can build upon and extend our theoretical results to improve the performance of BB-CCH and SAT-CPT, which could benefit from external symmetry-breaking tools.

Instance	$\|V(G)\|$	$\|E(G)\|$	*tww*	$t[s]$	Calls	Tri-twins	Filters Exceeded	Orbit	Quick	PCache	Reorder
Bcs01	48	176	7	17019	201 M	7.6 M	129 G	34	4	1.1 G	31 M
BN 113	50	719	8	12039	93 M	88 K	70 G	0	9	653 M	5.6 M
1brf	49	412	8	6521	61 M	1.2 M	36 G	0	6	412 M	2.5 M
Queen 10×10	100	1470	28	1415	1.8 M	30	2.8 G	85	0	5.2 M	134 K
Huck	74	301	3	973	8.3 M	1.8 M	3.3 G	0	680 K	40 M	12 M
Oesoca	39	67	≤ 3	> 21600	466 M	134 M	116 G	2	39 M	2.1 G	529 M

Table 3: Difficult instances for BB-CCH and how much which technique helped. The first five instances could not be solved using a SAT encoding, the last instance could only be solved using a SAT encoding. We use K for thousand, M for million, and G for billion.

Acknowledgments

We acknowledge the support from the Austrian Science Fund (FWF), projects P32441, P36420, and W1255; and from the Vienna Science and Technology Fund (WWTF), project ICT19-065.

References

[Ahn *et al.*, 2022] Jungho Ahn, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. Bounds for the twin-width of graphs. *SIAM J. Discrete Math.*, 36(3):2352–2366, 2022.

[Bailleux and Boufkhad, 2003] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer Verlag, 2003.

[Balabán and Hlinený, 2021] Jakub Balabán and Petr Hlinený. Twin-width is linear in the poset width. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 6:1–6:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Balabán *et al.*, 2022] Jakub Balabán, Petr Hlinený, and Jan Jedelský. Twin-width and transductions of proper k-mixed-thin graphs. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2022.

[Bergé *et al.*, 2022] Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Biere *et al.*, 2020] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[Bonnet *et al.*, 2020] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 601–612. IEEE, 2020. Full version appeared in the *J. ACM* 69(1): 3:1-3:46 (2022).

[Bonnet *et al.*, 2021a] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1977–1996. SIAM, 2021.

[Bonnet *et al.*, 2021b] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Bonnet *et al.*, 2022a] Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé. Twin-width VIII: delineation and win-wins. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Bonnet *et al.*, 2022b] Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé. Twin-width VIII: delineation and win-wins. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Bonnet *et al.*, 2022c] Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Torunczyk. Twin-width IV: ordered graphs and matrices. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 924–937. ACM, 2022.

[Bonnet *et al.*, 2022d] Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé. Twin-width V: linear minors, modular counting, and matrix multiplication. *CoRR*, abs/2209.12023, 2022.

[Bonnet *et al.*, 2022e] Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1036–1056. SIAM, 2022.

[Bonnet *et al.*, 2022f] Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *Algorithmica*, 84(11):3300–3337, 2022.

[Dreier *et al.*, 2022] Jan Dreier, Jakub Gajarský, Yiting Jiang, Patrice Ossona de Mendez, and Jean-Florent Ray-

mond. Twin-width and generalized coloring numbers. *Discrete Math.*, 345(3):112746, 2022.

[Gajarský *et al.*, 2022] Jakub Gajarský, Michal Pilipczuk, and Szymon Torunczyk. Stable graphs of bounded twin-width. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 39:1–39:12. ACM, 2022.

[Ganian *et al.*, 2022] Robert Ganian, Filip Pokrývka, Andre Schidler, Kirill Simonov, and Stefan Szeider. Weighted model counting with twin-width. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Jacob and Pilipczuk, 2022] Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 287–299. Springer Verlag, 2022.

[Martins *et al.*, 2014] Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental cardinality constraints for maxsat. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming*, pages 531–548, Cham, 2014. Springer International Publishing.

[McKay and Piperno, 2014] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, {II}. *J. Symbolic Comput.*, 60(0):94–112, 2014.

[Pilipczuk *et al.*, 2022] Michal Pilipczuk, Marek Sokolowski, and Anna Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 52:1–52:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Schidler and Szeider, 2022] André Schidler and Stefan Szeider. A SAT approach to twin-width. In Cynthia A. Phillips and Bettina Speckmann, editors, *Proceedings of ALENEX 2022, the 24nd SIAM Symposium on Algorithm Engineering and Experiments*, pages 67–77. SIAM, 2022.

[Sinz, 2005] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer Verlag, 2005.