A Caching Branch & Bound Approach to Twin-Width


André Schidler
Algorithms & Complexity Group, TU Wien, Austria
https://www.ac.tuwien.ac.at/people/aschidler/


Stefan Szeider
Algorithms & Complexity Group, TU Wien, Austria
https://www.ac.tuwien.ac.at/people/szeider/

Abstract

Twin-width is a recently proposed width-measure for graphs. It has gained much attention because of its solving power and generality. Bounded twin-width subsumes other prominent structural restrictions such as bounded treewidth and bounded rank-width. Determining the twin-width of a graph is NP-hard itself, already for twin-width 4.

In this paper we describe our branch & bound approach that determines the exact twin-width of a graph. We compute a sequence of contractions witnessing the twin-width and reduce the search space using several theoretical insights. We further speed up our search by caching results and can thereby skip whole sub-trees in our search tree.

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases Dummy keyword

Digital Object Identifier 10.4230/LIPIcs.PACE.2023.23

Supplementary Material https://doi.org/10.5281/zenodo.8033459, https://github.com/ASchidler/tww-bb

Funding *André Schidler*: Austrian Science Fund (FWF), projects P32441, P36420, and W1255; Vienna Science and Technology Fund (WWTF), project ICT19-065

1 Introduction

Twin-width, originally proposed by Bonnet et al. [13], has become a popular width measure for graphs, due to its theoretical properties. Twin-width generalizes many previously known graph classes for which important problems are tractable, including first-order model checking. Most notably, graphs of bounded treewidth, bounded clique-width, and planar graphs all have bounded twin-width [13]; 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 [12, 7, 8, 9, 9, 10, 11, 5, 6, 14, 4, 15, 19, 3, 2, 17, 1, 16].

Despite its popularity, few methods are known to compute the twin-width of a graph. Until the PACE Challenge 2023[1] the only method known to determine the exact twin-width was based on a SAT encoding [20]. In this paper, we describe our submission to the exact track, which is discussed in more detail in our related work [21].

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

[1] https://pacechallenge.org/2023/

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

A *contraction sequence* is a sequence of contractions $u_1 \to v_1, \ldots, u_k \to v_k$, such that for all $1 \leq i < j \leq k$ holds that $u_i \neq u_k$, i.e., contracted vertices cannot be contracted again. Whenever $k = |V(G)| - 1$ the contraction sequence is a *full contraction sequence*, as the graph after applying all contractions consists of only a single vertex.

We use the shorthand $G_0 = G$ and for any $i > 0$ we define $G_i = G_{i-1}^{u_i \to v_i}$. The *width* of a contraction sequence in the maximum red degree $\max_{0<i<|V(G)|} \max_{v \in V(G_i)} N_{G,R}(v)$. The twin-width of a trigraph G, denoted $tww(G)$, is the minimum width over all possible contraction sequences. Recognizing graphs of twin-width 4 is NP-complete [4].

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 Algorithm

We give the general idea of our algorithm, for more details, we refer the reader to our original paper [21].

We use a branch & bound approach that iterates over all possible contraction sequences by branching on the contractions. Conceptually, the algorithm is a recursive algorithm that chooses a new contraction in every recursive call. E.g., in the first call the algorithm chooses $u_1 \to v_1$, in the next recursive call $u_2 \to v_2$, until eventually the algorithm chose $|V| - 1$

contractions, which form a full contraction sequence. The upper bound is given by the discovered full contraction sequence with the lowest width. Initially, the upper bound is infinite.

Whenever we choose a contraction, we only consider *viable* contractions: contractions that do not use already contracted vertices and where the contraction does not reach the upper bound. Hence, whenever we reach a full contraction sequence, we improve the upper bound. Furthermore, we process the viable contractions ordered by the maximum red degree of the trigraph after the contraction, and, in case of a tie, by the number of new red edges. This increases the runtime of computing one full contraction sequence to $O(n^3)$ instead of $O(n)$, and the memory requirement is in $O(n^3)$. The actual bounds significantly depend on the actual number of viable contractions. The ordering usually finds contraction sequences of much lower width, and, hence, reduces the number of viable contractions.

Even for small graphs, the number of possible contraction sequences is too large to iterate over all in reasonable time. While the above method already filters out many sub-optimal contraction sequences, further methods are required to discover the twin-width of a graph within the 30 minutes allowed by the competition.

One simple but effective improvement can be applied whenever we find twins during our search for viable contractions. In this case, we can immediately contract the twins and ignore all other contractions, without risking a sub-optimal result.

We also apply several further improvements that are based on the idea of identifying isomorphic trigraphs. Given two trigraphs G_i and G'_i, derived from two contraction sequences $u_1 \to v_1, \dots, u_k \to v_k$ and $u'_1 \to v'_1, \dots, u'_k \to v'_k$. Whenever G_i and G'_i are isomorphic, then $tww(G_i) = tww(G'_i)$. Hence, whenever we know $tww(G_i)$, we can use that result instead of computing $tww(G'_i)$.

We can use this idea to limit the number of initial contractions we need to consider, i.e., the candidates for $u_1 \to v_1$. We use the tool *nauty* [18] to determine the graph's automorphisms and orbits. We then pick one vertex from each orbit and only consider contractions using at least one vertex from this subset. The trigraphs derived from any other contraction would be isomorphic to one of these contractions and can be skipped.

The most powerful method used by our algorithm is *partition-based caching*, where we use a key-value store that uses the partition of a trigraph as the key. Whenever we perform a contraction and obtain a trigraph $G_{i>0}$, we check if $P(G_i)$ is present in the cache. If yes, we use this result, otherwise, we store $tww(G_i)$ after we computed it. While the cache requires a lot of memory, it allows us to skip whole sub-trees of the search tree. We ensure that the cache does not grow too large by evicting entries based on a least-recently used policy.

The last improvement to our algorithm revolves around the observation that the order of contractions generally matters, but often does not. Hence, given a contraction sequence of width d, we can often re-order the contractions and still maintain the width of d. Hence, whenever the current contraction sequence $u_1 \to v_1, \dots, u_k \to v_k$ can be reordered such that $P(G_i)$ is in the cache, we can use the cached result. We use this in the following way. Let d be the observed maximum red degree for the current contraction sequence $u_1 \to v_1, \dots, u_k \to v_k$. After we perform the contraction $u_k \to v_k$, we try to put $u_k \to v_k$ as the $(k-1)$-th contraction, then as the $(k-2)$-nd contraction, and so on, always leaving the other elements in the same order. We stop whenever the maximum red degree of the modified contraction sequence exceeds d. After each reordering, we check if the partition trigraph, derived from performing all the contractions up to and including $u_k \to v_k$, is in the cache. Using some theoretical insights, we can perform this check efficiently at the price of not considering all possible reorderings.

References

1. 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. doi:10.1137/21M1452834.
2. 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. doi:10.4230/LIPIcs.IPEC.2021.6.
3. 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. doi:10.1007/978-3-031-15914-5_4.
4. 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. doi:10.4230/LIPIcs.ICALP.2022.18.
5. 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. doi:10.4230/LIPIcs.IPEC.2022.9.
6. 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. doi:10.4230/LIPIcs.IPEC.2022.9.
7. 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.
8. 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. doi:10.4230/LIPIcs.ICALP.2021.35.
9. 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. doi:10.1145/3519935.3520037.
10. 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. arXiv:2209.12023, doi:10.48550/arXiv.2209.12023.
11. 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. doi:10.1137/1.9781611977073.45.

12 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *Algorithmica*, 84(11):3300–3337, 2022. doi:10.1007/s00453-022-00965-5.

13 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).

14 Jan Dreier, Jakub Gajarský, Yiting Jiang, Patrice Ossona de Mendez, and Jean-Florent Raymond. Twin-width and generalized coloring numbers. *Discrete Math.*, 345(3):112746, 2022. doi:10.1016/j.disc.2021.112746.

15 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. doi:10.1145/3531130.3533356.

16 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. doi:10.4230/LIPIcs.SAT.2022.15.

17 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. doi:10.1007/978-3-031-15914-5_21.

18 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, {II}. *J. Symbolic Comput.*, 60(0):94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

19 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. doi:10.4230/LIPIcs.STACS.2022.52.

20 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. doi:10.1137/1.9781611977042.6.

21 André Schidler and Stefan Szeider. Computing twin-width with sat and branch & bound. In *IJCAI 2023*. ijcai.org, 2023. to appear.