PACE Solver Description: Touiouidth

Gaétan Berthe
LIRMM, CNRS, Université de Montpellier, France

Yoann Coudert–Osmont
Université de Lorraine, CNRS, Inria, LORIA, France

Alexander Dobler
Algorithms and Complexity Group, TU Wien, Austria

Laure Morelle
LIRMM, CNRS, Université de Montpellier, France

Amadeus Reinald
LIRMM, CNRS, Université de Montpellier, France

Mathis Rocton
Algorithms and Complexity Group, TU Wien, Austria

Abstract

We describe Touiouidth, a twin-width solver for the exact-track of the 2023 PACE Challenge: Twin Width. Our solver is based on a simple branch and bound algorithm with search space reductions and is implemented in C++.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Twinwidth, Pace Challenge

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.38

Supplementary Material *Software (source code)*: `https://doi.org/10.5281/zenodo.8027195`

Funding *Gaétan Berthe*: Supported by the ANR project ELIT (ANR-20-CE48-0008-01).
Alexander Dobler: Supported by the Vienna Science and Technology Fund (WWTF) under grant [10.47379/ICT19035].
Laure Morelle: Supported by the ANR project ELIT (ANR-20-CE48-0008-01) and the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027).
Amadeus Reinald: Supported by the ANR project DIGRAPHS (ANR-19-CE48-0013).
Mathis Rocton: Supported by the European Union's Horizon 2020 research and innovation COFUND programme (LogiCS@TUWien, grant agreement No 101034440), and the FWF Science Fund (FWF project Y1329).

1 Introduction

Twin-width is a graph parameter introduced in 2020 by Bonnet, Kim, Thomassé and Watrigant [3]. The PACE 2023 Exact Challenge asks to compute the exact twin-width of a dataset of 200 graphs. It is known that deciding whether a graph has twin-width 4 is NP-complete [1], and no approximation algorithms are known. On the exact front, there exists a polynomial-time algorithm for deciding whether a given graph has twin-width 1 [2], and the complexity remains open for twin-width 2 and 3. The exact computation of twin-width has already been tackled using SAT solver methods in [4].

A *trigraph* $G = (V, E_b, E_r)$ consists of a vertex set V, a set of black edges E_b, and a set of red edges E_r. All trigraphs considered in this paper are undirected, finite, and without loops or multiple edges (red or black). Let $N_b^G(v) = \{x \mid \{v, x\} \in E_b\}$ be the open black neighbourhood of v in G, and $N_r^G(v) = \{x \mid \{v, x\} \in E_r\}$ be its open red neighbourhood. The *red degree* of $v \in V$ is $\delta_r(v) = |\{e \in E_r \mid v \in e\}|$. The *red degree* of G is $\max_{v \in V} \delta_r(v)$.

A *contraction* in a trigraph G consists of merging any two vertices u and v into a single vertex w, and updating the edges of G as follows. Every vertex $x \in N_r^G(u) \cup N_r^G(v) \cup (N_b^G(u) \Delta N_b^G(v))$ is linked to w by a red edge, and every vertex $x \in N_b(u) \cap N_b(v)$ is linked to w by a black edge. All other edges in the graph (those not incident to u or v) remain unchanged. A *contraction sequence* $(G_1, \ldots, G_{|V|})$ for G is a sequence of contractions starting from $G_1 = G$ with no red edge and ending in $G_{|V|}$ consisting of a single vertex, where G_{i+1} is obtained from G_i by performing a contraction. A *k-contraction sequence* is such a sequence in which all graphs G_i have red degree at most k. Then, the *twin-width* of the graph is the minimal k for which there exists a k-contraction sequence.

In the following sections we describe our algorithm finding such an optimal contraction sequence. Section 2 describes a simple reduction rule. Section 3 describes the branch and bound algorithm that is used in several parts of our final algorithm, which is described in Section 5.

2 Reduction Rule

The following reduction rule is a generalisation of twins for trigraphs and is applied in all parts of our algorithms.

▶ **Reduction Rule 1.** *If there exist two vertices $u, v \in V$ such that*

- $N_b^G(u) \subseteq N_b^G(v)$, *and*
- $(N_r^G(v) \cup N_b^G(v)) \subseteq (N_r^G(u) \cup \{u, v\})$,

then contract u and v.

The correctness of this reduction rule is immediate: Let G' be the graph after contracting u and v. Then $G' = G[V \setminus \{v\}]$ and the twin-width of induced trigraphs can only decrease [3]. Note that u and v are not symmetric in the above description of the reduction rule. But the rule as it is presented above is easier to implement.

3 Main Branch and Bound Algorithm

The main building block of our algorithm is a branch and bound algorithm that tries all pairs of possible contractions in each search tree node. In the following, we present a few optimizations made in our solver.

Search space reduction. Consider a search tree node x of our branch and bound algorithm. Assume we contract vertices u and v in G, and recurse into child node y of the search tree. We do not have to try contracting u and v in the search trees rooted at sibling nodes of y until some contraction "touches" $N_G^2(u)$ or $N_G^2(v)$ where N_G^2 is the closed radius-two neighbourhood in G (considering black and red edges) (see Algorithm 1). We store these so-called forbidden contractions (set S in Algorithm 1) using a global vector of sets.

Algorithm 1 A sketch of the recursive branch and bound algorithm that avoids symmetries. The input is a trigraph G and a set S of *forbidden contractions*. Both are updated during the search tree.

```
1 SearchTree(G, S):
2     for (u, v) ∈ V(G) × V(G) do
3         if u < v ∧ (u, v) ∉ S then
4             S' ← S \ ((N²_G(u) ∪ N²_G(v)) × V(G)) \ (V(G) × (N²_G(u) ∪ N²_G(v)));
5             SearchTree(contract(G, u, v), S');
6             S ← S ∪ {(u, v), (v, u)};
```

Branching order. To guide the order in which we explore the search tree induced by our branch and bound algorithm, we consider all possible contractions in a search tree node in a given order. Namely, for $u, v \in V$, let $\Delta(u, v) = |((N_b(u) \Delta N_b(v)) \cup (N_r(u) \cup N_r(v))) \setminus \{u, v\}|$. Then we contract u_1 and v_1 before u_2 and v_2 if $\Delta(u_1, v_1) < \Delta(u_2, v_2)$. We sort all pairs using bucket sort, using one bucket for each $k = \Delta(u_1, v_1)$. This is faster than sorting all pairs using standard sorting algorithms when k is small. We observed that our initial upper bounds are small for most instances, so this improved the overall runtime of our algorithm.

Optimisations. We list further optimisations that we apply during the branch and bound process.

- If the current trigraph has at most 64 vertices then a vector of unsigned long long is used to store the current adjacency lists (for red and black edges). If bit j for index i in this data structure is one then this indicates the presence of an edge. This allows for faster set operations on neighbourhoods of vertices.
- At each search tree node, contract twins based on Reduction Rule 1. To find these twins, we do not have to try all pairs of vertices, but only those that are in the neighbourhood of contractions.
- We use a global upper bound and local/global lower bounds to leave infeasible parts of the search tree as early as possible.

4 Upper Bounds

We apply two heuristics that compute contraction sequences yielding us upper bounds. The first, called Heuristic1, iteratively contracts two vertices u, v such that $\Delta(u, v)$ is minimal among all pairs of vertices of the current graph. The second heuristic, called Heuristic2, uses randomness and can be executed several times. Given the current graph and an upper bound T, it contracts a random pair of vertices among all pairs u, v such that, after the contraction of u and v, the red degree of the trigraph does not exceed $T - 1$. If Heuristic2 is able to find a contraction sequence with twin-width less than T then this twin-width can be used as new upper bound for the next iteration.

5 The Complete Algorithm

The complete algorithm consists of three phases: computing a lower bound, computing an upper bound, and then using the branch and bound algorithm. It is run on each connected component of the graph, and each phase is given a time limit as indicated below.

Lower bound (15 minutes). A lower bound is computed based on the twin-width of induced subgraphs of G. We start with $|V|$ induced subgraphs, each containing one distinct vertex of the graph. Then, for each graph G_i, we iteratively add a vertex v; v has to be connected to G_i and v's minimum symmetric difference of neighbourhoods (in G) with vertices in G_i is maximal. After adding a vertex, the twin-width of the newly obtained induced subgraph is computed, and the lower bound is updated.

Upper bound heuristics (5 minutes). Next, Heuristic1 is ran once, and Heuristic2 is called multiple times in the remaining time. In a lot of instances, Heuristic2 is able to match the lower bound. So we are done and do not have to go to the next stage of the algorithm.

Exact Branch and Bound (10 minutes). The remaining time the exact branch and bound algorithm is applied to the input trigraph G. The upper bound obtained from the previous algorithm is used to leave infeasible parts of the search tree early. If a contraction sequence with twin-width matching the lower bound from the first phase is found, we report that contraction sequence as the solution. Otherwise, if the twin-width is larger than that lower bound, the complete search space (without infeasible parts due to the upper bound and without symmetries) is explored to find the contraction sequence with the smallest twin-width.

References

1 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding Twin-Width at Most 4 Is NP-Complete. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *Proc. International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.18.

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

3 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

4 André Schidler and Stefan Szeider. A SAT approach to twin-width. In Cynthia A. Phillips and Bettina Speckmann, editors, *Proc. Symposium on Algorithm Engineering and Experiments (ALENEX 2022)*, pages 67–77. SIAM, 2022. doi:10.1137/1.9781611977042.6.