Description of the Dino solver for PACE 2023

Alan Cabral Trindade Prado
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brazil

Emanuel Juliano Morais Silva
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brazil

Guilherme de Castro Mendes Gomes
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brazil

Kaio Henrique Masse Vieira
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brazil

Laila Melo Vaz Lopes
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brazil

Abstract

This article provides a concise overview of our solver, Dino, which was submitted in the Exact Track challenge of the Parameterized Algorithms and Computational Experiments 2023 (PACE2023). The objective of this challenge is to develop a deterministic approach for determining the twin-width [3] of a graph. Our submission primarily builds upon the SAT formulation presented in [6]. In addition to the relative SAT encoding, we have also incorporated an algorithm to derive a modular-like decomposition of the input graph [5], as well as a method for computing the twin-width of trees.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases twin-width, SAT, Cographs, Modular Decomposition, PACE 2023

Supplementary Material Source code repository: `https://github.com/gcmgomes/pace2023`

1 Brief Description of the Algorithm

Let G denote the input graph, and let $tww(G)$ represent its twin-width. Our solution begins by decomposing the input graph into smaller graphs, as described in [5]. Subsequently, an optimal contraction sequence is computed for each graph in the decomposition, and these sequences are merged (recomposed) to form an optimal contraction sequence for G. The decomposition and recomposition steps are detailed in Sections 1.1 and 1.2, respectively.

To compute an optimal contraction sequence, we first determine whether the given graph is a tree. If it is, we can leverage the observation 1 to compute its twin-width, as the decomposition step never produces a cograph with more than one vertex.

If the graph is not a tree, we rely on the relative SAT encoding proposed in [6]. Our implementation employs the CaDiCaL solver [2] to evaluate the necessary SAT instances. Each SAT encoding represents a Boolean formula dependent on G and d, which is true if and only if $tww(G) \leq d$. Therefore, to compute the actual twin-width, we perform a binary search.

As an upper bound for the binary search, we utilize the width of the contraction sequence obtained by always selecting the contraction that minimizes the red-degree in the subsequent iteration. Similarly, to achieve a lower bound, we consider all possible first contractions and select the one resulting in the minimum red-degree.

It is also important to note that implementing cardinality constraints required the utilization of *totalizer* constraints [1], as suggested in [6].

1.1 Decompose

The decomposition step is utilized as a preprocessing step. To effectively handle larger instances, the role of the decomposition step is to replace the input graph by a series of smaller graphs. Subsequently, during the recomposition step, the contraction sequence obtained from the smaller graphs are merged.

We employ a preprocessing step similar to the one outlined in [6]. In polynomial time, we decompose a graph G into a collection denoted as $\boldsymbol{module(G)}$, consisting of induced subgraphs of G. This decomposition ensures that $tww(\mathrm{G})$ is equal to $\max_{H \in module(G)} tww(H)$. It is important to note that, unlike decomposing a graph into prime graphs, our approach focuses on achieving a decomposition solely into *modules*.

We adhere to the established definitions. A *module* of a graph G refers to a nonempty set $M \subseteq V(G)$ satisfying the condition that for any $x, y \in M$ and $z \in V(G) \setminus M$, the edge $xy \in E(G)$ if and only if the edge $yz \in E(G)$. A graph is considered *prime* if all of its modules are trivial.

During the decomposition phase, given an instance graph G, we construct a modular-like decomposition tree $MLD(G)$ and a set $module(G)$ recursively, following these steps:

(1) If G is disconnected, then $module(G)$ is the union of the sets $module(C)$ for all connected components C of G. Additionally, for each $MLD(C)$, an edge is connected from the current node of $MLD(G)$.
(2) If the complement graph $\overline{G}$ is disconnected, the set $module(G)$ is defined as the union of the sets $module(\overline{C})$ for all connected components $\overline{C}$ of $\overline{G}$. Furthermore, an edge is connected from the current node of $MLD(G)$ to each $MLD(\overline{C})$.
(3) If a non-trivial module decomposition of G is found, then $module(G)$ is defined as the union of $G \setminus P$ and the sets $module(G[M])$ for all non-trivial modules $M \in P$, where P represents the modular partition that has been found.

By solely implementing the first and second steps, we can achieve a complete decomposition of cographs, effectively creating the co-tree [4] in polynomial time. This co-tree is subsequently utilized to obtain a contraction sequence with width 0.

In the third step, we employ the `Modular Partition` algorithm as described by Habib [5]. This algorithm takes a partition P of the vertex set V of a graph G as input and returns the coarsest modular partition Q that is smaller than P. As the initial partition, we select an arbitrary vertex v and utilize the partition $P = \{\{v\}, N(v), \overline{N}(v)\}$. In a subsequent section of the paper, Habib explains how to find the *maximal modular partition* of G by constructing $MD(G)$. However, we have not implemented this part and therefore cannot guarantee that our partition is maximal with respect to G.

As we cannot guarantee that our partition is maximal, we implemented a heuristic step with the objective of creating the smallest $G_{/P}$. Instead of using an arbitrary vertex v for the initial partition, we iterate over a limited number of vertices and keep the partition resulting in the smallest $G_{/P}$. Note that it is sufficient to utilize any modular partition in the decomposition step. As a result, this heuristic step does not impact the exactness of our solution.

1.2 Recompose

During the recomposition process, we are provided with a modular-like decomposition tree $MLDT$ and a set of contraction sequences. Each leaf node of the $MLDT$ has a corresponding contraction sequence, and there are additional contraction sequences associated with internal

nodes representing the quotient graphs $G_{/P}$. The output of the `recompose` function is the contraction sequence of the original graph.

This function operates recursively. To compute the recomposition of a subtree, it initially calculates the contraction sequence for all children of the current node. If the current node is of type **(1)** or **(2)** from the contraction steps [1.1], the children are contracted in an arbitrary order. However, if the current node is of type **(3)**, we follow the contraction sequence of $G_{/P}$.

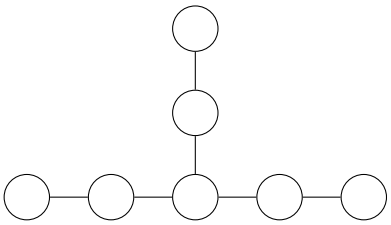
1.3 Trees

As described previously, the first step of the solver is to evaluate whether the given graph is a tree and, if so, compute its contraction sequence. The proof of the following observation explicitizes a polynomial algorithm to compute an optimal sequence of contractions for a given tree in polynomial time.

▶ **Observation 1.** *Let T be a tree. Suppose T is not a cograph. Then, $tww(T) = 1$ if and only if T is a caterpillar and, otherwise, $tww(T) = 2$.*

Proof. To begin with, we will show that is always possible to define a sequence of contractions in a tree that yields twin-width at most two. First, we choose an arbitrary vertex as a root for the tree. Then, we repeat the following process until we're left only with this root: while there are any twin leaves (leaves with the same parent), merge them. Once there are no twin leaves, merge any leaf with its parent and return to the previous step. It is easy to see that it is not possible to have a vertex with red-degree three when a tree is contracted using this strategy.

The same procedure above can be used to show that the twin-width of a caterpillar is at most one, with the modification that the vertex used to root the tree is one of the extremes of its central path. Given that we are not considering cographs, it must be the case that the twin-width of T is equal to one if it is a caterpillar. It remains to show that the only trees with twin-width one are caterpillars. We will use a proof by contraposition. First, note that caterpillars are exactly the trees that do not have the following as an induced subgraph:



It is easy to compute all possible vertex contraction sequences and see that this graph has twin-width two. [3] shows that the twin-width of a graph must be greater than or equal to the twin-width of any induced subgraph of it and, therefore, a tree that is not a caterpillar must have twin-width two.

◀

Finally, we note that it is possible to implement a variation of the procedure described above in linear time. Given an arbitrary tree, we first search for a vertex that is an extremity of a path which has length corresponding to the diameter of the graph. Subsequently a simple depth-first search can be used to contract the leaves of the tree and compute a valid and optimal contraction sequence. This procedure does not need to distinguish whether a tree is indeed a caterpillar or not.

References

1 Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming – CP 2003*, pages 108–122, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

2 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.

3 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *CoRR*, abs/2004.14789, 2020. URL: https://arxiv.org/abs/2004.14789, arXiv:2004.14789.

4 D.G. Corneil, H. Lerchs, and L.Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981. URL: https://www.sciencedirect.com/science/article/pii/0166218X81900135, doi:https://doi.org/10.1016/0166-218X(81)90013-5.

5 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010. URL: https://www.sciencedirect.com/science/article/pii/S157401371000002X, doi:https://doi.org/10.1016/j.cosrev.2010.01.001.

6 André Schidler and Stefan Szeider. A SAT approach to twin-width. *CoRR*, abs/2110.06146, 2021. URL: https://arxiv.org/abs/2110.06146, arXiv:2110.06146.