UAIC_Twin_Width: An Exact Twin-Width Algorithm

Andrei Arhire
Alexandru Ioan Cuza University of Iași, Romania

Matei Chiriac
Alexandru Ioan Cuza University of Iași, Romania

Radu Timofte
ETH Zürich, Switzerland
University of Würzburg, Germany
Alexandru Ioan Cuza University of Iași, Romania

Abstract

Twin-width is a graph parameter that provides an intuitive measure of the similarity between a graph and a co-graph [1]. To understand this concept, let's consider a tri-graph initially consisting of only black edges. We repeatedly apply an operation until only one vertex remains in the graph: two vertices are merged into a new vertex, and the edges are updated based on specific rules. The neighborhood of the new vertex is formed by combining the neighborhoods of the contracted vertices. Within this neighborhood union, an edge is considered black if it appears as a black edge in both original neighborhoods; otherwise, it becomes red. The twin-width is defined as the highest red degree encountered at any vertex during the contraction steps that minimize this degree.

In this short paper, we provide an overview of the main techniques utilized in the exact twin-width algorithm, which we have submitted as the UAIC_Twin_Width solver for the exact track of the 2023 PACE challenge.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases twin-width, dynamic programming, local search

Supplementary Material The source code is available on Zenodo (10.5281/zenodo.8010483) and GitHub (https://github.com/AndreiiArhire/PACE2023).

1 Preliminaries

Consider a trigraph $G = (V, B, R)$ consisting of a set of vertices V, black edges B, and red edges R. Here, B and R are subsets of $\binom{V}{2}$, and they do not have any common elements ($B \cap R = \emptyset$). Without any specific indication, a graph $H = (V, E)$ is treated as the trigraph $H' = (V, E, \emptyset)$.

In this context, we define the red degree of a vertex as the number of incident red edges connected to that vertex. When two vertices are compressed, a new vertex is formed by considering the edges associated with the original vertices, and subsequently, the original vertices are removed from the graph.

Formally, the union of vertices x and y, with B_x, B_y, R_x, and R_y representing the sets of incident red and black edges, gives rise to a vertex z with the following definitions:

- A vertex $u \in R_z$ if $u \in B_x \cup B_y \cup R_x \cup R_y \setminus B_x \cap B_y$.
- A vertex $u \in B_z$ if $u \in B_x \cap B_y$.

As per the requirements of the challenge format, the solver assigns the same label to z as that of x. By following this process of reducing the graph to a single vertex through these operations, a sequence of $|V| - 1$ pairs of vertices is generated. The width of such a sequence

is defined as the maximum red degree encountered at any vertex during the contraction process.

The twin-width represents the minimum width achievable through a sequence of contractions that reduces the graph.

2 Solver summary

The solver can be divided into four stages:

1. Contract vertices that do not create any red edges when contracted.
2. Find an upper bound for the entire graph using a hill climbing algorithm.
3. Determine a lower bound for each connected component by using dynamic programming on multiple sub-graphs.
4. Calculate the twin-width using dynamic programming. Evaluate components in descending order based on their number of vertices (expected twin-width). A component can yield a suboptimal solution as long as another component has a twin-width greater than or equal to its width.

The core of the solver relies on a dynamic programming algorithm that iterates over subsets of potential solutions. A solution sequence can be visualized as a tree, where x is the parent of y if the pair (x, y) is part of the sequence. Thus, the remaining vertex in the graph serves as the root. A partial solution can be seen as a set of trees.

A state in dynamic programming holds the following information:

- The sequence of contractions leading to the current solution.
- The width of the current solution.
- The set of trees determined by the sequence of contractions.
- The initial state starts with an empty set, and the final state includes the twin-width value and the corresponding sequence. Transitioning between states is achieved by appending a pair of vertices to the partial solution.

To optimize the algorithm, the following observations are utilized:

- All pairs used for state transitions consist of either adjacent vertices or vertices with at least one neighbor in common.
- If there exists a pair of vertices whose contraction does not create any new red edges, it is optimal to contract that pair (as the vertices are twins).
- To avoid ending up in non-optimal states multiple times, partial solutions are computed for l pairs first, followed by partial solutions for $l + 1$ pairs, where $l < |V|$.
- When comparing a state with t trees $\alpha_1, \ldots, \alpha_t$ to a state with t trees $\beta_1, \ldots, \beta_t$, where α_i and β_i (for $1 \leq i \leq t$) have the same vertices but in different configurations, only the state with the lower width is retained. Thus, the removed state is guaranteed to be incapable of obtaining a better solution than the one kept.
- Prior to running the algorithm, a low-width solution (upper bound) and a lower bound value are computed.
- A state with a sequence width greater than or equal to the upper bound is excluded.
- A state is excluded if the graph obtained by applying the sequence of contractions has a width lower or equal to the width of the state's sequence. For each state, a heuristic algorithm is used to find an upper-bound solution for the generated graph. If the upper bound is lower than or equal to the current width, appending the heuristic solution to the current sequence will result in a sequence with the same width.
- A state with a width lower than the lower bound does not need to invoke the heuristic algorithm.

3 Upper Bounds

An upper bound is calculated using two algorithms, namely the hill climbing algorithm (H_a) and the greedy algorithm (G_a), although they are used at different stages. On the entire graph, H_a is utilized to determine an upper bound within a few minutes. Meanwhile, as part of the decision-making process, G_a is used to determine whether a state should be excluded from the set of partial solutions.

G_a constructs the sequence by gradually appending pairs of vertices to it, aiming to minimize the maximum red degree in the graph obtained after contracting the vertices.

H_a begins with a primary solution generated by G_a with a width of w. Let d_i, $1 \leq i \leq |V| - 1$, represent the maximum red degree of a vertex during the first i contractions. Since w represents the width, $d_{|V|-1} = w$ at the very least. Consider $1 \leq p \leq |V| - 1$, where $d_i = d_{i+1} = \cdots = d_{|V|-1}$ and $d_{i-1} < w$.

H_a explores multiple derived solutions and retains the one with the highest value of p, which indicates the solution that remains of width w for as long as possible, preferably transitioning to width $w - 1$.

Let $(x_1, y_1), \cdots, (x_p, y_p), \cdots, (x_{|V|-1}, y_{|V|-1})$ be a solution, where p is as described above. A potentially improved solution is obtained through the following steps:

- Randomly select integers a and b from the range 1 to p, and randomly choose vertices u and v such that $u \neq y_i, 1 \leq i \leq a$ and $v \neq y_i, 1 \leq i \leq b$ (where u is present in the graph after the a-th operation and v is present after the b-th operation).
- Update x_a to be equal to u.
- Swap y_b with v, starting from line p.
- Initiate G_a from the partial solution consisting of the first p (or fewer) pairs.

Once a specific number of derived solutions have been computed, the best-evaluated solution is designated as the primary solution. The complexity of G_a is $\mathcal{O}(|V|^3 \cdot |E|)$, while H_a has a complexity of $\mathcal{O}(|V|^3 \cdot |E| \cdot T)$, where T represents the number of local searches performed.

In the case of public tests that are small enough for G_a to execute rapidly, H_a achieved the optimal answer in approximately one-fourth to one-third of the tests.

4 Lower Bounds

The twin-width of a graph is always greater than or equal to the twin-width of any of its sub-graphs. Exploiting this concept, we can determine a lower bound by utilizing the exact algorithm on various sub-graphs. When applied to public tests, this technique swiftly identifies the optimal solution for tests where the twin-width is at most three.

For small tests with a twin-width no greater than five, the upper bound often matches the lower bound. Consequently, it is unnecessary to execute the dynamic programming algorithm on the entire graph.

References

1 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width i: tractable fo model checking. *ACM Journal of the ACM (JACM)*, 69(1):1–46, 2021.