METATWW: Finding Small-Width Contraction Sequences for PACE 2023

William Bille Meyling ✉
University of Copenhagen, Denmark

Abstract

In the heuristic track of the PACE (Parameterized Algorithms and Computational Experiments) 2023 challenge the aim is to find contraction sequences of small width. Our solver, `METATWW`, uses a two-phase approach. In phase one it finds an initial contraction sequence by contracting pairs of vertices where the size of the symmetric difference between their neighbourhoods is small. In phase two we iteratively move contractions around within specific intervals of the contraction sequence in order to reduce the width of the sequence.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

Keywords and phrases Metaheuristic, Similar neighbourhoods, Twinwidth, Algorithm engineering

Supplementary Material `https://gitlab.com/willthbill/pace2023-twinwidth`; `https://doi.org/10.5281/zenodo.8045541`

Acknowledgements Thanks to Mikkel Abrahamsen and Kasper Skov Johansen for some great discussions.

1 Introduction

The topic of the PACE 2023 challenge (`https://pacechallenge.org/2023`) is twinwidth. The twinwidth of a graph measures its distance to a co-graph. This metric can be computed using **contraction sequences**. A **contraction** merges two vertices, a and b, into a single vertex, c, such that the neighborhood of c is the union of the open neighborhoods of a and b[1]. Before any contraction is performed we say that all edges are **black**, but after performing contractions edges may become **red**. Suppose, after a contraction has been performed, an edge, $\{c, d\}$, is created. If d is only a neighbor of one of a and b, then $\{c, d\}$ will become red. If d is a neighbor of both a and b and at least one of the edges, $\{a, d\}$ and $\{b, d\}$, is already red, then $\{c, d\}$ will become red. Otherwise $\{c, d\}$ becomes black. The **red-degree** of a vertex is the number of red incident edges to the vertex. The **width** of a contraction sequence is the maximum red-degree of any vertex at any point throughout the sequence of contractions. The twinwidth of a graph is equal to the smallest possible width of a contraction sequence that turns the graph into a single-vertex graph. It is, however, computationally hard to exactly compute the twinwidth[1] and therefore the focus of the heuristic track of PACE 2023 is to just find contraction sequences of small width.

Our solver, `METATWW`, consists of two phases – in phase one a heuristic produces an initial solution and in phase two a metaheuristic iteratively improves the solution. In phase one we find the initial contraction sequence of small width by looking at the size of the symmetric difference of the neighbourhoods of the vertices being contracted. In case this heuristic is too slow we instead simply look at the sizes of the neighbourhoods of the vertices being contracted. The contractions are found one at a time by choosing a pair of vertices that minimizes these sizes. In phase two we iteratively reduce the width of the contraction sequence by randomly

[1] before merging, any edges between a and b are removed.

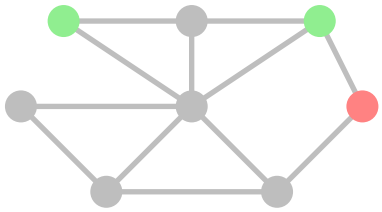

Figure 1 Illustration of the *slow heuristic*. The green pair of vertices is a pair, where the size of the symmetric difference (the red vertex) is minimum (size is 1).

moving contractions around within intervals of the sequence, such that we are maintaining a valid contraction sequence[2] at all times. We run this solver independently on each connected component of the input graph.

We believe that looking at the symmetric difference of neighbourhoods can work well as a heuristic, since it is related to the number of red-edges that will be created by a contraction. We unfortunately did not have time to optimize other heuristics that potentially could outperform symmetric difference of neighbourhoods. We would have also liked to develop a more sophisticated version of our metaheuristic. Lastly, we missed out on very good solutions to some graph-instances in the PACE challenge, because of poor estimates of the running time of the heuristics in phase one. We think this solver has much potential for further improvement.

2 Phase one: The Initial Contraction Sequence

In the first phase of the solver we find an initial contraction sequence. We will *choose* the contractions one at a time and append them to the sequence. For a given graph we use one of the following two heuristics to construct the whole contraction sequence.

- *(fast heuristic)* We order the vertices in non-decreasing order of degree, and *choose* the first two vertices.
- *(slow heuristic)* Each edge will be associated with a value given by the size of symmetric difference between the open neighbourhoods of the endpoints of the edge. As we will discuss, we allow the values to not always be exact. We *choose* the two endpoints of the edge with lowest value. Figure 1 illustrates the *slow heuristic*.

Ideally, for the *slow heuristic*, we would have liked to consider all pairs of vertices of distance at most two. However, even when only considering edge contractions, it is not easy to efficiently maintain the values associated with each edge. After contracting an edge, $\{a, b\}$, that results in the creation of a new vertex, c, we have to recompute the values of all edges incident to a neighbour of c. The value of an edge not incident to c can decrease by at most two. This allows for the recomputation to be done in $\tilde{O}(1)$ time per edge, but due to the quadratic behavior of this situation this is still too slow. Instead we will use the following strategy after each contraction. Here each vertex will have an associated counter initially equal to zero.

1. Recompute values for edges incident to c.
2. Mark the neighbours of c, and increase their counters by one.

[2] valid meaning that for all contractions the two two contracted vertices existed right before the contraction.

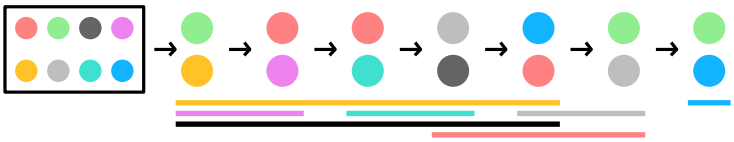

Figure 2 Illustration of the intervals used by the metaheuristic for a given contraction sequence. The black box contains the vertices of the input graph. The new vertex created by a contraction has the color of the top vertex. The interval associated with a contraction has the color of the bottom vertex.

3. We will order the marked vertices in non-increasing order by their counter. Then for the first 1% of marked vertices we unmark them, set their counters to zero and recompute the values of their incident edges.

The third step is motivated by the fact that edge values can decrease by at most two, and thus recomputation does not seem important until many changes are observed.

3 Phase two: Iteratively Improving the Solution

Given a contraction sequence we wish to modify it to decrease its width. Let p_v be the index in the contraction sequence where vertex v is created (p_v is 0 if v was part of the input graph), and let n_v be the index where v is contracted with another vertex (n_v is $|V|$ if v is the last vertex remaining after all contractions). Let the contraction at index i be given by the ordered pair of vertices $C_i = (a, b)$, which are contracted into vertex c, and consider the interval, $I_i = \{\max(p_a, p_b) + 1, \ldots, n_c - 1\}$. If we move C_i to any other index in I_i (such that C_i happens at a different time), then the contraction sequence is still valid. These intervals are illustrated in figure 2. Based on this, our very basic metaheuristic is the following.

- Find the first contraction, C_i, which resulted in a red-degree equal to the width of the whole contraction sequence.
- Move C_i to a uniformly random index taken from interval I_i, but only if this move reduces the width of the contraction sequence.

We repeat the above until we run out of time (5 minutes in total for the challenge). However, based on observations it seems that improvements happen fast in the beginning and then slows down. In the context of this metaheuristic it looks like it often finds a contraction sequence of minimal width.

4 Algorithmic Optimizations

In previous sections details are left out about how the solver is made efficient. Here are the most important algorithmic optimizations.

- We use a sparse bit-set to represent the list of neighbours of a vertex. Updating takes logarithmic-time, but counting the size of the intersection, union or symmetric difference of two neighbourhoods can be done in linear time (and even faster in practice especially if the graph is dense).
- When contracting two vertices the order does not matter. Thus, we choose to always insert the neighbours of the vertex with smallest degree into the neighbourhood of the vertex with largest degree. This way, each vertex will be inserted at most a logarithmic number of times.

References

1 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is np-complete. *CoRR*, abs/2112.08953, 2021. URL: `https://arxiv.org/abs/2112.08953`, `arXiv:2112.08953`.