Twinwidth Approximation via Breadth-First-Search

Jonathan Balasingham
University of Liverpool, United Kingdom https://www.liverpool.ac.uk/

Sebastian Wild
University of Liverpool, United Kingdom https://www.liverpool.ac.uk/

Viktor Zamaraev
University of Liverpool, United Kingdom https://www.liverpool.ac.uk/

Abstract

This note describes the approach of our submission to the PACE 2023 twin-width competition, heuristic track. We perform a modular decomposition and contract nodes within modules by a local search that contracts neighbors with minimal symmetric difference.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Twinwidth, Graph Theory

Digital Object Identifier 10.4230/LIPIcs.PACE.2023.23

Funding This work was supported by an Impact-Accelerator Grant by the School of Electrical Engineering, Electronics and Computer Science of University of Liverpool and the NeST (Network Sciences and Technologies) EEECS School initiative of University of Liverpool.

1 Introduction

An (undirected, simple) graph is a pair (V, E) where V is the set of vertices $V = \{v_1, \ldots, v_n\}$ and E is the set of edges represented by unordered pairs of vertices, $E = \{(v_i, v_k) \ldots\}$. Given a graph $G = (V, E)$, a contraction sequence is an ordered list of vertex pairs such that each pair (v_i, v_j) results in the contraction of these nodes into a single node represented by v_i. The graph hence, has one less node at every step of the sequence. Since v_j has been remove, it cannot appear at any point after in the contraction sequence. We will denote by G_t, the graph after t contractions; $G_0 = G$ and $G_{n-1} = K_1$, the single-vertex graph. After contraction (v_i, v_j), the new neighborhood of v_i is now $N_{t+1}[v_i] = N_t[v_i] \cup N_t[v_j]$. Of the edges in $N_{t+1}[v_i]$, the ones belonging to $N_t[v_i] \cap N_t[v_j]$ will remain black if they were previously. If these edges belong to those in the symmetric difference of the nodes' neighborhoods, $N_t[v_i] \triangle N_t[v_j]$, then they will become red. Any edge that may have existed between v_i and v_j is no longer present, regardless of color. The *(maximum red) degree* of a contraction sequence is the maximum red degree that occurs in the graph during any one of the sequence steps; the *twin-width* [2] of a graph G, $\operatorname{tww}(G)$, then is the minimal degree of any contraction for G. The purpose of our solver is to quickly find a low-degree contraction sequence.

2 Solver Approach

It is obvious that exact twins (vertices with the same neighborhood apart from themselves) can always be contracted at no cost. We generalize this twin-removal reduction by performing a *modular decomposition* [3] on the input graph G using the implementation by Fabien de Montgolfier available at `https://github.com/vbraun/graph-modular-decomposition`. By definition, there is always an optimal degree contraction sequence that contracts all modules of a graph first, hence this is a (relatively) inexpensive step that helps exploit structure in the graph by giving us a set of induced subgraphs, which can be solved independently. The

Listing 1 Traversal of Modular Decomposition Tree

```
def traverse($G$, $T$, node):
    for $c$ in children($T$, node)
        if type(node) == LEAF
            $M$ := siblings($T$, $c$);
            BFSSolve($G[M]$);
        traverse($G$, $T$, $c$);

$T$ := MDTree($G$)
traverse($G$, $T$, root($T$))
```

Listing 2 Breadth-First-Search Solver

```
def BFSSolve(G):
    $v_c$ = vertex with smallest id in $G$
    while $|V(G)|$ > 1
        scores := []
        if size(neighbors($v_c$)) < 1
            $v_c$ = find_next_vertex($G$, $v_c$);
            continue
        for $v_i$ in neighbors($G$, $v_c$)
            $s$ := symmetric_difference($G$, $v_c$, $v_i$);
            scores.append(($|s|$, $v_i$));
        sort(scores);
        contract($G$, $v_c$, scores[0][1]);

def find_next_vertex($G$, $v_c$):
    return vertex $u$ from first $\rho$ vertices in $G$ that
           minimizes symmetric difference of neighborhoods of $u$ and $v_c$
```

output of the modular decomposition algorithm is a modular decomposition tree. To obtain the modules, we traverse the tree and find the leaves of the tree in a post-order depth-first search. The pseudocode for the traversal algorithm is shown in Listing 1. If G is prime, the original graph is passed to the next step. Solving each of the modules of G first, allows us to bound the max red degree of the overall sequence to the maximum red degree which occurs in any of the modules.

Within the prime graphs, we use a greedy local-search heuristic. Given a module of G, we select an initial starting node v_i at random. From v_i, we look at each of its neighbors $v_j \in N[v_i]$ and compute the symmetric difference $N[v_i] \triangle N[v_j]$. The sizes $|N[v_i] \triangle N[v_j]|$ are treated as scores to determine which of the neighbors will be contracted with v_i. The neighbor with the smallest symmetric difference with v_i will be contracted. This process continues until the entire module has been contracted into a single node. Once all modules are solved, the result is the original graph with each module substituted with a single node. This process is applied on this graph.

Should it be found that the graph is disconnected, and thus cannot be reached through neighbors, the next starting node is determined by finding a node within the graph with the smallest symmetric difference. Here, we also limit the number of nodes searched by introducing another parameter ρ (we use $\rho = 1000$). Once the vertex with the smallest symmetric difference is found, this vertex is contracted with v_i and the process continues as described before. The pseudocode for the algorithm is shown in Listing 2.

3 Noteworthy Implementation Details

In our implementation, there are a few things that must be noted. First, when computing the symmetric difference between two nodes' neighborhoods, we create a cache for the current node v_i. By doing this, we save time by not having to recompute symmetric differences for neighboring nodes that have been checked. Given the time constraint, this cache is necessary in order to be able to complete medium to large sized graph. By keeping track of already computed symmetric differences, the complexity of the solver is linear in the number of nodes and edges. This is not without cost, however. It is possible for a contraction to invalidate the value of the symmetric difference in the cache. In these cases, its possible for the an undesirable node to be contracted with the current node. Fortunately, this is only a feature that is used to make sure computations are faster and can be removed if time is not as limited.

Second, the solver produces the best results when edges are more sparse, regardless of the number of nodes. The extreme of this case is when the graph forms a chain (seen in example 1 of the tiny test set [1]). This in turn leads to the weakness of the solver on dense graphs. Further, if the nodes which are most desirable to be contracted are not connected, the solver will not contract them (unless the current node is isolated). An example of this can be seen in graph 10 of the tiny set, where nearly all members of the contraction sequence consist of disconnected nodes [1]. Finally, when the graph is large, the modular decomposition algorithm will not be able to be completed within the time constraint. We have determined this threshold to be approximately $150,000$ vertices.

References

1 Max Bannach and Sebastian Berndt. Pace 2023: Tiny test set. `https://pacechallenge.org/2023/tiny-set.pdf`.

2 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width i: tractable fo model checking. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 601–612, 2020. `doi:10.1109/FOCS46700.2020.00062`.

3 Michel Habib, Fabien De Montgolfier, and Christophe Paul. A simple linear-time modular decomposition algorithm for graphs, using order extension. In *Algorithm Theory-SWAT 2004: 9th Scandinavian Workshop on Algorithm Theory, Humlebæk, Denmark, July 8-10, 2004. Proceedings 9*, pages 187–198. Springer, 2004.