PACE Solver Description: Bute-Plus: A Bottom-Up Exact Solver for Treedepth

James Trimble
School of Computing Science, University of Glasgow, Scotland, UK
j.trimble.1@research.gla.ac.uk

Abstract

This note introduces *Bute-Plus*, an exact solver for the treedepth problem. The core of the solver is a positive-instance driven dynamic program that constructs an elimination tree of minimum depth in a bottom-up fashion. Three features greatly improve the algorithm's run time. The first of these is a specialised trie data structure. The second is a domination rule. The third is a heuristic presolve step can quickly find a treedepth decomposition of optimal depth for many instances.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Algorithm design techniques

Keywords and phrases Treedepth, Elimination Tree, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.34

Related Version `https://arxiv.org/abs/2006.09912`

Supplementary Material DOI of code submitted to PACE Challenge: `https://doi.org/10.5281/zenodo.3881441`. The latest version of the code can be found on GitHub: `https://github.com/jamestrimble/pace2020-treedepth-solvers`.

Funding *James Trimble*: This work was supported by the Engineering and Physical Sciences Research Council (grant number EP/R513222/1).

Acknowledgements I would like to thank the PACE 2020 program committee and the optil.io team for a well-organised and enjoyable contest.

1 Introduction

A treedepth decomposition of graph G is a rooted forest F, such that if G has edge $\{u, v\}$ then either u is an ancestor of v or v is an ancestor of u in F. The treedepth problem is to determine, for a given graph G, the minimum depth of a treedepth decomposition of G, where depth is defined as the maximum number of vertices on a root-leaf path.

An *elimination tree* of a connected graph G is a special type of treedepth decomposition, defined recursively as follows. If G has a single vertex, its elimination tree equals G. Otherwise, let v be a vertex in G and let F be a forest consisting of an elimination tree for each component of $G - v$. Then an elimination tree of G is formed by making v the parent of every root of F.

For every connected graph G, there exists an elimination tree whose depth equals the treedepth of G ([5], chapter 6). To solve the treedepth problem, it is therefore sufficient to find an elimination tree of minimum depth. That is the approach taken by the Bute-Plus solver, which this paper introduces. The solver uses a positive-instance driven dynamic programming algorithm, which seeks sets of vertices that induce low-treedepth subgraphs of the input graph. Three additional features improve the performance of the algorithm: a specialised trie data structure, a domination rule based on a rule by Ganian et al. [2], and a heuristic presolver that quickly finds an optimal solution for many of the PACE Challenge instances.

The author submitted two other exact solvers to the PACE Challenge: Bute (which is Bute-Plus without the heuristic presolve step) and Bute-Plus-Plus (which spends additional time on the heuristic presolve and has a minor modification to the trie data structure). An earlier algorithm by the author [9], which constructs a treedepth decomposition from the top down, is very memory-efficient but is typically much slower than Bute-Plus.

2 A brief description of the algorithm

This section presents an outline of the Bute-Plus algorithm. We assume that the vertex set of a graph G, denoted $V(G)$, contains only integers. The neighbourhood of vertex v is denoted by $N(v)$. For a set of vertices S, $N(S)$ denotes the set of vertices that are not in S but are adjacent to some member of S.

The algorithm takes as input a connected graph G and returns the treedepth of G along with a treedepth decomposition of that depth. The optimisation problem is solved as a sequence of decision problems. The solver attempts to find an elimination tree of depth 1, then of depth 2, and so on until it is successful. (Typically, the higher-numbered decision problems are by far the most time consuming.)

For the decision problem of whether an elimination tree of depth k exists, the algorithm works downwards for $i = k, \ldots, 1$, finding all subsets S of $V(G)$ such that (1) S induces a subgraph of G with treedepth no greater than $k - i + 1$, (2) the neighbourhood of S has fewer than i vertices, and (3) the subgraph of G induced by S is connected. This collection of sets of vertices is called $\mathcal{S}_i^k$; it includes the vertex set of every subtree whose root is at depth i of an elimination tree of depth k of G.

The algorithm uses a positive-instance driven (PID) [7] approach to constructing the $\mathcal{S}_i^k$: rather than generating all subsets of $V(G)$ and checking if each one satisifies the three required properties, the elements of $\mathcal{S}_i^k$ are generated by joining together elements of $\mathcal{S}_{i+1}^k$. To be more precise, sets in $\mathcal{S}_i^k$ are constructed in two ways; a sketch of these follows. The first is simply by choosing vertices with a sufficiently small neighbourhood (since clearly each of these induces a connected subgraph of treedepth 1). The second is by finding a nonempty sub-collection $\mathcal{S} \subseteq \mathcal{S}_{i+1}^k$ and a vertex v satisfying the following conditions. The elements of $\mathcal{S}$ must be pairwise disjoint, and moreover there must not be an edge between vertices in any two distinct members of $\mathcal{S}$. Furthermore, v must have an edge to at least one vertex in each member of $\mathcal{S}$. These conditions guarantee that the set $\bigcup \mathcal{S} \cup \{v\}$ induces a connected subgraph of G of that admits an elimination tree with root v of depth no more than $k - i + 1$.

It is easy to verify using the definition of $\mathcal{S}_i^k$ that an instance of the decision problem is satisfiable if and only if $\mathcal{S}_1^k$ is non-empty (in which case $\mathcal{S}_1^k$ will have $V(G)$ as its only element). A small amount of extra bookkeeping allows the solver to output an optimal elimination tree.

Bute-Plus is not the first PID algorithm for treedepth. Bannach and Berndt [1] present a PID framework for computing a range of graph parameters including treedepth, treewidth, and pathwidth. Although their paper describes the family of algorithms in terms of a game theoretic characterisation of each problem, their algorithm for treedepth has a similar overall approach to that of Bute-Plus: both algorithms build up sets of vertices by combining one or more existing sets with a root vertex. Bannach and Berndt use a queue when combining sets whereas Bute-Plus uses a stack; a second difference is that the algorithm of Bannach and Berndt is not restricted to finding only elimination trees. The framework of Bannach and Berndt generalises a PID algorithm for treewidth by Tamaki which won the exact treewidth track of PACE 2016;[1] a second PID algorithm for treewidth by Tamaki [7] performed strongly in PACE 2017.

[1] `https://github.com/TCS-Meiji/treewidth-exact`

3 Improvements to the algorithm

The Bute-Plus solver has three additional features which greatly reduce run time on many instances. Two of these – a trie data structure and a domination rule – are described in the following two subsections. The third feature is a heuristic solver which is run for the first minute with the hope of finding a treedepth decomposition of optimal depth; this uses the Tweed-Plus solver which was an entry by the author in the heuristic track of PACE 2020 and is described in its own paper in this volume.

3.1 Trie data structure

Recall from Section 2 that the algorithm generates sets in the collection $\mathcal{S}_i^k$ by finding a subset of $\mathcal{S}_{i+1}^k$ along with a vertex v that together satisfy certain properties. For some of the PACE Challenge instances, $\mathcal{S}_{i+1}^k$ can contain millions of sets, and the task of finding appropriate subsets of the collection becomes intractable without a specialised data structure.

Bute-Plus's data structure supports two operations. The first is to add a $(S, N(S))$ pair – a set of vertices and its neighbourhood – to the collection. The second is a query operation which takes a set of vertices Q and an integer i. This returns all sets S in the collection such that both (1) $|N(S) \cup N(Q)| < i$ and (2) $(Q \cup N(Q)) \cap S = \emptyset$.

The data structure is implemented as a trie. When $(S, N(S))$ is inserted, $N(S)$ is sorted in ascending order and viewed as a string over the alphabet $V(G)$, then added to the trie. This approach has been used for the similar problem of superset queries several times in the past, for example in Savnik's Set-Trie [6].

To sketch the query operation: the algorithm performs a depth-first traversal of the trie, backtracking when it becomes clear that no value in the subtree is acceptable. For efficiency, each node of the trie stores the intersection of $N(S)$ values in the subtree rooted at that node; this idea is from a data structure for superset queries posted on Stack Overflow by Ben Tilly [8].

The task carried out by Bute-Plus's data structure is similar to the task of the *block sieve* designed by Tamaki for a PID treewidth solver [7]. Although both data structures are based on tries, their designs differ in several respects; for example, the block sieve data structure comprises a collection of tries rather than just one.

3.2 Domination rule

As discussed in Section 1, to find a minimum-depth treedepth decomposition it is sufficient to restrict attention to elimination trees. We can speed the algorithm up further by placing additional restrictions on acceptable elimination trees, if it can be shown that at least one tree in the restricted class has optimal depth.

For this purpose, the Bute algorithm uses a domination-breaking rule that extends a rule by Ganian et al. [2]. For distinct vertices v, w, we say that v *dominates* w if either of the following two conditions holds: (1) $N(v) \setminus \{w\} \supset N(w) \setminus \{v\}$; (2) $N(v) \setminus \{w\} = N(w) \setminus \{v\}$ and $w < v$. It is always possible to construct an elimination tree of minimum depth such that no vertex dominates any of its ancestors.

This rule allows us to further restrict each collection $\mathcal{S}_i^k$ to include only sets of vertices S such that no vertex in S dominates any member of $N(S)$.

4 Implementation details

The Bute-Plus solver is written in C. Sets of vertices are stored using bitsets; code from Nauty 2.6r12 [4] is used for the bitset data structure.[2] The Tweed-Plus heuristic presolver also uses code from Nauty for for the random number generator, and uses Metis 5.1.0 [3] to find nested dissection orderings.[3]

References

1 Max Bannach and Sebastian Berndt. Positive-instance driven dynamic programming for graph searching. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2019. doi:10.1007/978-3-030-24766-9_4.

2 Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. SAT-encodings for treecut width and treedepth. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019.*, pages 117–129. SIAM, 2019. doi:10.1137/1.9781611975499.10.

3 George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998. doi:10.1137/S1064827595287997.

4 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

5 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

6 Iztok Savnik. Index data structure for fast subset and superset queries. In *Availability, Reliability, and Security in Information Systems and HCI - IFIP WG 8.4, 8.9, TC 5 International Cross-Domain Conference, CD-ARES 2013, Regensburg, Germany, September 2-6, 2013. Proceedings*, volume 8127 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2013. doi:10.1007/978-3-642-40511-2_10.

7 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, 2019. doi:10.1007/s10878-018-0353-z.

8 Ben Tilly. Fast data structure for finding strict subsets (from a given list). Stack Overflow. Version: 2011-06-29. URL: https://stackoverflow.com/a/6514445/3347737.

9 James Trimble. An algorithm for the exact treedepth problem. In *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy*, volume 160 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SEA.2020.19.

[2] Nauty is available at http://pallini.di.uniroma1.it/

[3] Metis is available at http://glaros.dtc.umn.edu/gkhome/metis/metis/overview