Memoizing Treedepth Brancher

Philip de Bruin
Utrecht University, Dept. Information and Computing Sciences & Dept. Mathematics
p.debruin@students.uu.nl

Erik Jan van Leeuwen
Utrecht University, Dept. Information and Computing Sciences
e.j.vanleeuwen@uu.nl

Abstract

In this paper, we describe the ingredients to our PACE 2020 submission, the Memoizing Treedepth Brancher, for the exact computation of treedepth. At a high level, the program combines a standard dynamic program for treedepth with a branch-and-bound approach. Furthermore, the program follows a positive-instance driven approach and implements an algorithm for the special case of trees.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Treedepth, exact algorithms, branch-and-bound, positive-instance driven

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Supplementary Material The implementation of the algorithm discussed here is freely available [1].

1 Introduction

The *treedepth* $\mathbf{td}(G)$ of a graph G is the smallest possible depth of any tree T such that $V(G) = V(T)$ and for any $(u, v) \in E(G)$, u is an ancestor or descendant of v in T. Such a tree is called a *treedepth decomposition*. Many NP-hard and even some polynomial-time solvable graph problems can be solved more efficiently if the treepdepth is bounded. This also led to an investigation into the computation of treedepth. For that purpose, often the following recursive definition of Nesetril and Ossona de Mendez [4] for treedepth is used:

$$\mathbf{td}(G) = \begin{cases} 1 & \text{if } \left|V(G)\right| = 1 \\ 1 + \min\limits_{v \in V(G)} \max\limits_{H \in \mathcal{C}(G \setminus v)} \mathbf{td}(H) & \text{otherwise.} \end{cases} \tag{1}$$

As described in Fomin et al. [2], this definition naturally leads to a dynamic programming approach. The table stores, for each $S \subseteq V(G)$, the treedepth of $G[S]$. Then each entry can be rapidly computed using Equation 1, in order of smallest set S to largest. This results in a running time of $O^*(2^n)$ and equal space usage.

The observation that started our work is that by applying memoization, particularly when combined with a branch-and-bound approach, we might significantly lower both the practical running time and space usage. Below, we describe these optimizations as well as optimizations for (sub)graphs that are trees. The final solver, the MemoizingTreedepthBrancher [1], solves 26 of the 100 public instances of the PACE 2020 challenge.

2 Trees and Paths

The treedepth of a tree or path can be computed in polynomial time [3, 5]. Hence, our solver detects that the graph is a tree or a path and then uses the corresponding polynomial-time algorithm. The motivation to address these special cases came from the observation that several graphs in the public instances of the challenge have path- or tree-like "attachments".

For a path P on n vertices, Perarnau and Serra [5] proved that $\mathbf{td}(P) = \lfloor \log n \rfloor + 1$. For a tree, no such simple formula exists. We implemented the $O(n \log n)$-time algorithm by Iyer et al. [3] to compute an optimal node ranking of a tree (this algorithm is simpler than the known linear-time algorithm), because the smallest maximum rank of any such ranking is equal to the treedepth. We also designed and implemented a recursive algorithm for trees that converts the node ranking to a treedepth decomposition in $O(\mathbf{td}(G) \cdot n)$ time.

3 Branch-and-bound Approach and Positive-Instance Driven

We implemented a branch-and-bound approach to speed up the recursive algorithm. Note that Case 2 of Equation 1 branches on the vertex v to remove from the graph. Then the best bound for any previous branch or of any supergraph of G', where G' is the graph in the current step of the recursion, forms an effective upper bound. To help form a strong upper bound early in the algorithm, we branch on vertices in non-increasing order of their degree.

As a lower bound, we follow an observation of Nesetril and Ossona de Mendez and use $\lfloor \log k \rfloor + 1$, where k is the height of a depth-first search (DFS) tree of G'. Note that $\mathbf{td}(H) \leq \mathbf{td}(G')$ for any minor H of G' [4]. If the height of the DFS tree is k, we can form a path of k vertices which is a minor of G'. Thus $\mathbf{td}(G')$ is lower bounded by $\lfloor \log k \rfloor + 1$. Note that the recursive definition of Equation 1 requires computing the connected components of the new graph, for which we use DFS, so that this bound can be readily and easily included.

We also made our program positive-instance driven (PID), inspired by Tamaki's highly successful approach for the treewidth problem [6]. As a starting upper bound, we used $\lfloor \log n \rfloor + 1$. In each of the following iteration, we multiplied this upper bound by $\frac{3}{2}$.

Finally, we report on some tests of the impact of the different ingredients of our algorithm on its performance. As an example, the table below shows the running times of different variations of our algorithm when solving Instance 25 of the public instances for PACE 2020:

Implementation	Average over 3 runs [s]
Without branch and bound.	826.786
With branch and bound, but without sorting vertices.	26.702
With branch and bound and sorting vertices.	9.586
With branch and bound and PID.	8.982

This table and some further testing suggest that the inclusion of branch and bound and sorting by degree had the strongest impact on performance, while the additional impact of PID was negligible (and sometimes adverse). For future work, we propose further testing of this suggestion as well as evaluating the faster-than-2^n algorithm by Fomin et al. [2].

References

1 Philip de Bruin. PhiliPdB/treedepth-exact: Submission for PACE, May 2020. URL: `https://github.com/PhiliPdB/treedepth-exact`, `doi:10.5281/zenodo.3866006`.

2 Fedor V. Fomin, Archontia C. Giannopoulou, and Michał Pilipczuk. Computing tree-depth faster than 2^n. *Algorithmica*, 73(1):202–216, 2015.

3 Ananth V. Iyer, H. Donald Ratliff, and G. Vijayan. Optimal node ranking of trees. *Information Processing Letters*, 28(5):225–229, 1988.

4 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.

5 G. Perarnau and O. Serra. On the tree-depth of random graphs. *Discrete Applied Mathematics*, 168:119–126, 2014.

6 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019.