Computing Exact Treedepth via Minimal Separators

Zijian Xu
The University of Tokyo, Japan
xuzijian@ms.k.u-tokyo.ac.jp

Dejun Mao
The University of Tokyo, Japan
maodejun001@is.s.u-tokyo.ac.jp

Vorapong Suppakitpaisarn
The University of Tokyo, Japan
vorapong@is.s.u-tokyo.ac.jp

Abstract

This is a description of team xuzijian629's treedepth solver submitted to PACE 2020. As we use top-down approach, we enumerate all possible minimal separators at each step. The enumeration is sped up by several novel pruning techniques and is based on our conjecture that we can always have an optimal decomposition without using separators with size larger than treewidth. The algorithm could solve 79 public instances at PACE 2020.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Treedepth, Minimal Separators, Experimental Algorithm

Digital Object Identifier 10.4230/LIPIcs.PACE.2020.0

1 Notations

In this paper, G denotes undirected unweighted graph. $V(G)$ or simply V denote the vertex set. We use n and m for the number of nodes and edges, respectively. The treedepth of G is expressed as $td(G)$. For a vertex set $S \subseteq V$, $G[S]$ is the subgraph induced by S. We use $G \backslash S$ for the graph obtained from G by removing S, that is, $G \backslash S = G[V \backslash S]$. S is called an a-b separator if $a, b \in V$ are not connected in $G \backslash S$. S is called a minimal a-b separator if S is an a-b separator and any proper subset of S is not an a-b separator. Finally, S is called a minimal separator if S is a minimal a-b separator for some $a, b \in V$.

2 Algorithm

We explain our main algorithm first, then discuss our pruning techniques and the preprocessing.

2.1 Main Algorithm

The recursive formula we use for computing treedepth is a variant of [2].

▶ **Theorem 1** (Treedepth computation via minimal separators [1]).

$$td(G) = \begin{cases} |V| & \textit{if } G \textit{ is a complete graph} \\ \min_{S \in \mathcal{S}} |S| + \max_{H \in \mathcal{C}(G \backslash S)} td(H) & \textit{otherwise} \end{cases}$$

where $\mathcal{S}$ denotes the set of all minimal separators of G, and $\mathcal{C}(G \backslash S)$ denotes the set of connected components in $G \backslash S$.

A graph may have exponential number of minimal separators, but they can be enumerated in $O(n^3m)$ per object [4]. We use our following conjecture to reduce the number of minimal separators that we have to enumerate.

▶ **Conjecture 2.** *Let $\mathcal{S}_{tw} = \{S \in \mathcal{S} : |S| \leq tw\}$, and let td' be a function obtained from td in Theorem 1 by replacing $\mathcal{S}$ with $\mathcal{S}_{tw}$. For all G, $td(G) = td'(G)$.*

2.2 Pruning Rules

Our solver handles the decision version of the treedepth problem. It computes $solve(G, k)$, checking if $td(G) \leq k$, for $k = 1, 2, \ldots$. When G is separated by a minimal separator S, $solve(G, k)$ recursively checks $solve(H, k - |S|)$ for each connected component $H \in \mathcal{C}(G \backslash S)$. Since $solve(H, k - |S|)$ returns `false` for most separators S, it is essential to obtain a good lower bound of treedepth and prune the search space. Two most effective lower bounds are degeneracy and path-length bound. Degeneracy can be computed in linear time [3] and often works as the most effective bound especially for small graphs. Path-length bound looks for a long path P in G and lower bound treedepth by $\lceil \log_2(|P| + 1) \rceil$, where $|P|$ is the number of nodes in P [5]. We also use some other lower bounds but they are not as effective as those mentioned above.

We observed that all existing lower bounds can effectively prune our search only when n is as small as 100. Both of them are weak for large graphs, and degeneracy is not scalable there. To cope with this issue, we introduce a novel pruning heuristic for large graphs below.

Pruning by Blocks

As a preprocess, we take various induced subgraphs of the input graph. These subgraphs are called blocks. Let $\texttt{Blocks}[i]$ be the collection of blocks with size i. For each i in ascending order, we compute the exact treedepth of all its elements and sort them in descending order of treedepth. In $solve(H, k - |S|)$, we scan each blocks from smaller i and then from larger treedepth, and if there is a block $B \subseteq H$ and $V(B) \cap S = \emptyset$ such that $td(B) + |S| > k$, we can immediately return `false`. In computing exact treedepth for $\texttt{Blocks}[i]$, $\texttt{Blocks}[j]$ for $j < i$ is used for pruning. The preprocess is terminated either if it finishes the computation for all blocks or after time limit of 600 seconds. This pruning was so effective that it allow us to solve almost 20 public instances which we could not solve without it.

References

1. Jitender S Deogun, Ton Kloks, Dieter Kratsch, and Haiko Müller. On the vertex ranking problem for trapezoid, circular-arc and other graphs. *Discrete Applied Mathematics*, 98(1-2):39–63, 1999.
2. Dariusz Dereniowski and Adam Nadolski. Vertex rankings of chordal graphs and weighted trees. *Information Processing Letters*, 98(3):96–100, 2006.
3. David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983.
4. Ken Takata. Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph. *Discrete Applied Mathematics*, 158(15):1660–1667, 2010.
5. James Trimble. An algorithm for the exact treedepth problem. *arXiv preprint arXiv:2004.08959*, 2020.