PACE Solver Description: 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 a 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. Although we cannot theoretically guarantee that our algorithm based on the unproved conjecture can always give an optimal solution, it can give optimal solutions for all instances in our experiments. The algorithm solved 68 private instances and placed 5th in the competition.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Treedepth, Minimal Separators, Experimental Algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.31

Supplementary Material The solver submitted to the competition is available at `https://doi.org/10.5281/zenodo.3870624` and the repository is `https://github.com/xuzijian629/pace2020`.

1 Preliminaries

1.1 Notations

In this paper, G denotes undirected unweighted graph. V or $V(G)$ denote the vertex set. We use n and m for the number of nodes and edges, respectively. The treewidth and treedepth of G are expressed as $tw(G)$ and $td(G)$, respectively. $N(v)$ or $N_G(v)$ denote the open neighbors. 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, $G \backslash S = G[V \backslash S]$. Also, we use $\mathcal{C}(G)$ to denote the connected components of G. S is called an a-b separator if $a, b \in V$ are not connected in $G \backslash S$. An a-b separator is called minimal if none of its proper subset is an a-b separator. Finally, S is called a minimal separator if S is a minimal a-b separator for some $a, b \in V$.

1.2 Computing Treedepth via Minimal Separators

The recursive formula we use for computing treedepth is a variant of the following theorem.

▶ **Theorem 1** ([4]).

$$td(G) = \begin{cases} |V| & \text{if } G \text{ is complete} \\ \min_{S \in \Delta_G} \left(|S| + \max_{H \in \mathcal{C}(G \backslash S)} td(H) \right) & \text{otherwise} \end{cases} \tag{1}$$

where Δ_G is a collection of all minimal separators of G.

Algorithm 1 MinDegree and MinFill.

```
Require: graph $G$
Ensure: upper bound of treewidth $ub$
1: $H := G$
2: while $G$ is not empty do
3:     $v :=$ a vertex with minimum degree (MinDegree)/with minimum fill (MinFill)
4:     $F := \{(a, b) \mid a, b \in N_G(v) \text{ and } (a, b) \notin E(G)\}$
5:     add edges in $F$ to both $G$ and $H$
6:     remove $v$ from $G$
7: end while
8: assert $H$ is a chordal completion of $G$
9: $ub :=$ treewidth of $H$
```

The bottleneck of computing treedepth by Theorem 1 is the computation of Δ_G, since a graph may have an exponential number of minimal separators with respect to n.

To calculate an optimal treedepth decomposition from Equation (1), the authors begin by finding the set Δ_G. Then, for each minimal separator $S \in \Delta_G$ and for each connected component $H \in \mathcal{C}(G \backslash S)$, they recursively calculate $td(H)$. By that, they can obtain a set $S^* \in \arg\min_{S \in \Delta_G} \left(|S| + \max_{H \in \mathcal{C}(G \backslash S)} td(H) \right)$. An optimal treedepth decomposition obtained from the algorithm is a tree which:

1. the top of the tree is a simple path consisting of all nodes in S^*;
2. the bottom end of the simple path have several branches, each of the branches is connected to the root of an optimal treedepth decomposition for $H \in \mathcal{C}(G \backslash S)$, which can be computed recursively by the same algorithm.

2 Conjecture

In order to reduce the amount of the minimal separators that we have to enumerate, we conjectured that it suffices to enumerate minimal separators that have size at most treewidth. Formally,

▶ **Conjecture 2.** *Let Δ_G^p be a set of separators no larger than p, i.e. $\Delta_G^p := \{S \in \Delta : |S| \leq p\}$. Define $td^{(p)}(G)$ as follows:*

$$td^{(p)}(G) = \begin{cases} |V| & \text{if } G \text{ is complete} \\ \min_{S \in \Delta_G^p} \left(|S| + \max_{H \in \mathcal{C}(G \backslash S)} td(H) \right) & \text{otherwise} \end{cases} \tag{2}$$

Then, for any graph G and for any $p \geq tw(G)$, $td^{(p)}(G) = td(G)$.

It is known that we can efficiently calculate an upper bound of treewidth, denoted by $\overline{tw(G)}$ by taking the minimum of MinDegree heuristic and MinFill heuristic (Algorithm 1).

Then, if the conjecture is correct, we can replace the set S^* in the previous section with $\hat{S} \in \arg\min_{S \in \Delta_G^p} \left(|S| + \max_{H \in \mathcal{C}(G \backslash S)} td(H) \right)$. By that, our search space size is reduced from $|\Delta_G|$ to $\left| \Delta_G^{\overline{tw(G)}} \right|$, and we can significantly speed up the algorithm.

Correctness of the Conjecture and Solver

We cannot prove the correctness of Conjecture 2 for general graphs. Hence, we cannot theoretically guarantee that our solver based on the conjecture can always give an optimal solution. However, for all public and private instances we could solve, the solutions were optimal. Some theoretical aspects of this conjecture are discussed in [8].

3 Pruning Rules

In addition to the conjecture in the previous section, we speed up our solvers using several 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 $H \in \mathcal{C}(G \backslash S)$. If we have a method to efficiently calculate a lower bound of $td(H)$ for each H, we can immediately return `false` if the lower bound is larger than $k - |S|$. We can significantly prune the search space if the lower bound is tight. Therefore, we use several lower bounds of $td(H)$ in our solvers.

We compute the following six lower bounds from the first one. The first two lower bounds are usually not as effective as others, but the computation is much simpler.

Simple Bound. Consider a treedepth decomposition T such that it has k leaves and the depth is d. If T is a treedepth decomposition of G, then G may have at most $(k-1)(n-k/2)$ edges. The maximum case is obtained when all k leaves are at depth d and for $1 \leq i \leq d-1$, there is only one node at depth i. Therefore, we have the following lower bound for treedepth:

▶ **Proposition 3.** *Let k' be the smallest integer k such that $m \leq (k-1)(n-k/2)$, then $td(G) \geq k'$.*

Maximum Degree Bounds [7].

▶ **Proposition 4.** *Let $b > 0$ be the maximum degree of G. Then, $td(G) \geq lb(n)$, where*

$$lb(n) = \begin{cases} 0 & \textit{if } n = 0 \\ 1 + lb(\lceil (n-1)/b \rceil) & \textit{otherwise.} \end{cases}$$

Degeneracy Bound. Let G be a graph. The degeneracy of G is defined as the maximum of minimum degrees among all subgraphs. Degeneracy is a lower bound of treewidth [2] and can be computed in linear time [6] and often works as the most effective bound especially for small graphs. Since $td(G) \geq tw(G) + 1$, degeneracy plus one is a lower bound of treedepth.

Path-length Bound [7]. Let P be a path in G. Then, $td(G) \geq \lceil \log_2(|P| + 1) \rceil$, where $|P|$ is the number of nodes in the path P.

Contraction Degeneracy Bound [3, 5]. Contraction degeneracy is a stronger lower bound of treewidth, compared to degeneracy. However, since its computation takes time, we use this bound only when degeneracy is slightly smaller than k in $solve(G, k)$. The algorithm is described in Algorithm 2.

Pruning by Blocks. We observed that the above lower bounds can effectively prune our search only when n is as small as 100. For larger n, we introduce a novel pruning heuristic. As a preprocessing, we take various induced subgraphs of the input graph. These subgraphs are called blocks. Let `Blocks`$[i]$ be the collection of blocks with size i. For each i in ascending order, we compute the exact treedepth of the induced subgraphs and sort

Algorithm 2 Contraction Degeneracy Heuristic [3, 5].

```
Require: graph $G$
Ensure: lower bound of treewidth $lb$
1: $lb := 0$
2: while $G$ has more than one vertices do
3:     $v :=$ a vertex with minimum degree
4:     $lb \leftarrow \max(lb, degree(v))$
5:     $u :=$ a vertex in $N(v)$ such that $|N(u) \cap N(v)|$ is minimum
6:     contract $(u, v)$
7: end while
```

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 $td(B) > k - |S|$, we can immediately return `false`.

Block Selection in Preprocessing

To obtain various blocks, we need a quick randomized heuristic. We combine the following two partitioning heuristics to recursively decompose the input graph.

Partitioning Heuristic 1. Select two random vertices u and v in G and divide $V(G)$ into two groups, denoted U and V. A node v' is put in U if the shortest path length from v' to u is smaller than the shortest path length from v' to v. Otherwise v' is put in V. We then consider $G[U]$ and $G[V]$ as two blocks in the preprocessing process.

Partitioning Heuristic 2. A small vertex separator S is computed by [1]. We then consider each of the components of $G \backslash S$ as blocks.

To compute exact treedepth for the elements in `Blocks`$[i]$, `Blocks`$[j]$ for $j < i$ is used for pruning. The preprocessing is terminated either if it finishes the computation for all blocks or it exceeded the time limit for preprocessing. This pruning was so effective that it allows us to solve almost 20 public instances which we could not solve without it.

References

1 Haeder Y Althoby, Mohamed Didi Biha, and André Sesboüé. Exact and heuristic methods for the vertex separator problem. *Computers & Industrial Engineering*, 139:106135, 2020.

2 Hans L Bodlaender and Arie MCA Koster. Treewidth computations ii. lower bounds. *Information and Computation*, 209(7):1103–1119, 2011.

3 Hans L Bodlaender, Arie MCA Koster, and Thomas Wolle. Contraction and treewidth lower bounds. In *European Symposium on Algorithms*, pages 628–639. Springer, 2004.

4 Dariusz Dereniowski and Adam Nadolski. Vertex rankings of chordal graphs and weighted trees. *Information Processing Letters*, 98(3):96–100, 2006.

5 Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. *arXiv preprint*, 2012. `arXiv:1207.4109`.

6 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.

7 James Trimble. An algorithm for the exact treedepth problem. *arXiv preprint*, 2020. `arXiv:2004.08959`.

8 Zijian Xu and Vorapong Suppakitpaisarn. On the size of minimal separators for treedepth decomposition, 2020. `arXiv:2008.09822`.