PACE 2020 Exact Track: Solver Description

Narek Bojikian
Humboldt-Universität zu Berlin, Germany
bojikian@informatik.hu-berlin.de

Alexander van der Grinten
Humboldt-Universität zu Berlin, Germany
avdgrinten@hu-berlin.de

Falko Hegerfeld
Humboldt-Universität zu Berlin, Germany
hegerfeld@informatik.hu-berlin.de

Laurence Alec Kluge
Humboldt-Universität zu Berlin, Germany
klugelau@hu-berlin.de

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany
kratsch@informatik.hu-berlin.de

1 Solver Description

In this article we give a description of the treedepth solver that we submitted to the exact track of PACE 2020. The objective is to compute a treedepth decomposition of minimum depth. A *treedepth decomposition* of a graph $G = (V, E)$ is a rooted tree $T = (V, E_T)$ such that every edge of G connects a pair of nodes that have an ancestor-descendant relationship in T. The source code of our solver is available at `https://github.com/PACE-Challenge-Hu-Berlin/PACE-Challenge-2020` and [1]. We start by giving a brief overview of our solver and then describe the constituent parts in more detail.

Fundamentally our solver is a dynamic programming algorithm. Instead of using the naive dynamic programming algorithm which generates a subproblem for every connected induced subgraph and hence quickly becomes intractable, we pursue, inspired by the treewidth solver of Tamaki [4], a *positive-instance driven* approach: we construct so-called *feasible trees*, essentially treedepth decompositions of subgraphs, and glue them together to obtain ever larger feasible trees. We search for a treedepth decomposition that satisfies local optimality properties, similar to those of Fomin et al. [2]. Throughout the algorithm, we make crucial use of these properties to prune the number of generated feasible trees. One instantiation of these properties are *full separators*, i.e., separators so that each component is fully adjacent to the separator. We glue feasible trees together in such a way that a full separator appears. Another instantiation comes in the form of various *precedence rules*; these rules state that if a feasible tree contains some vertex u, then it must also contain a specific vertex v and thereby allow us to discard candidates where this is not the case.

1.1 Positive-Instance Driven Dynamic Programming

In this section we describe the objects generated by our dynamic programming more formally and how they are composed to yield larger objects. The concept of a full separator is fundamental for the composition of objects.

▶ **Definition 1.1.** Let S be a separator of $G = (V, E)$. We say that S is a *full separator* if $N(C) = S$ for every connected component C of $G - S$.

If $T = (V, E_T)$ is a rooted tree, we denote by T_v the subtree consisting of $v \in V$ and all of its descendants. The following lemma ensures that we do not make any errors by only considering full separators.

▶ **Lemma 1.2** ([2]). *Let G be a connected graph. There is a treedepth decomposition of G of minimum depth so that for every $v \in V$ the graph $G[T_v]$ is connected and the path from the root of T_v to the first node with multiple children is a full separator of $G[T_v]$.*

▶ **Definition 1.3.** Let $X \subseteq V$ be a vertex set and let $h \in \mathbb{N}$. We say that (X, h) is a *feasible tree* if there is a treedepth decomposition T of $G[X \cup N(X)]$ and some $x \in X$ so that T_x is a treedepth decomposition of $G[X]$ of depth h. If $|X| = h$, then (X, h) is called *trivial*.

To glue feasible trees together along a full separator, we define an intermediate object called *feasible forest* which is a collection of appropriate feasible trees.

▶ **Definition 1.4.** Let $\mathcal{F} \subseteq 2^V$ be a family of vertex sets and $h \in \mathbb{N}$. We say that $(\mathcal{F}, h)$ is a *feasible forest* if

- there is some $X \in \mathcal{F}$ such that (X, h) is a feasible tree and for every $X' \in \mathcal{F}$ there is a $h' \leq h$ so that (X', h') is a feasible tree,
- all $X \in \mathcal{F}$ are pairwise disjoint,
- X and $N(X')$ are disjoint for all $X \neq X' \in \mathcal{F}$,
- $\bigcap_{X \in \mathcal{F}} N(X)$ is non-empty.

The next lemma describes how we can compose feasible objects to obtain larger ones.

▶ **Lemma 1.5.** *There are two cases, depending on the number of feasible trees to be composed:*

- *Let $(\mathcal{F}, h)$ be a feasible forest with $|\mathcal{F}| > 1$. For any subset $S \subseteq \bigcap_{X \in \mathcal{F}} N(X)$, we obtain a feasible tree $(S \cup \bigcup_{X \in \mathcal{F}} X, h + |S|)$.*
- *Let (X, h) be a trivial feasible tree ($|\mathcal{F}| = 1$). For any $v \in N(X)$, we obtain a trivial feasible tree $(X \cup \{v\}, h + 1)$.*

The dynamic programming algorithm works as follows. We iterate over the maximum depth h starting at 1 until we find a feasible tree (V, h). In each iteration, we first construct the trivial feasible trees of height h. Afterwards, we construct feasible forests $(\mathcal{F}, h)$ from the previously computed feasible trees of height $h' \leq h$. From these feasible forests we obtain feasible trees as described in Lemma 1.5.

1.2 Preprocessing

Before running the dynamic programming algorithm, we apply some preprocessing steps. If the graph is not connected, then we consider each connected component separately. Universal vertices, i.e., vertices that are adjacent to all other vertices, are removed as they must be placed at the top of the treedepth decomposition. These steps are performed exhaustively until no connected component admits universal vertices.

1.3 Precedence Rules

We employ precedence rules to prune feasible objects that are not necessary to construct minimum-height decompositions. There are two types of precedence rules: *static* ones and *dynamic* ones. The static precedence rules are derived solely from the input graph, whereas the dynamic ones also take the subgraph induced by a feasible object into account. Intuitively, the static precedence rule discards feasible trees if their full separators contain suboptimal vertices, while the dynamic precedence rule forces more vertices into full separators.

Static Precedence Our static precedence rule makes use of the following statement:

▶ **Lemma 1.6.** *Let $v \neq w \in V$ be two vertices such that $N[w] \subseteq N[v]$. Then there is a minimum-height treedepth decomposition T such that w is a descendant of v in T.*

We construct a directed graph $P = (V, A)$ called *precedence graph* to exploit this property. A contains an arc (v, w) for each $v \neq w \in V$ such that $N[w] \subseteq N[v]$. Hence, arcs $(v, w) \in A$ in the precedence graph have essentially the following meaning: feasible trees (X, h) with $v \in X$ and $w \notin X$ do not need to be constructed. In other words, if a feasible tree contains v, then we can assume that it also contains w and given a vertex set $X \subseteq V$, we obtain the smallest meaningful subproblem containing X by determining the set of vertices that is reachable from X in the precedence graph.

We compute the strongly connected components of the static precedence graph and construct its condensation. In the base step of the dynamic programming algorithm, we only consider trees that correspond to nodes of the condensation that do not have any precedence-successors. We also check for precedence violations when forming feasible trees out of feasible forests $(\mathcal{F}, h)$. Let S be the full separator considered in this operation. If there is a vertex $v \in S$ and a precedence-successor w of v such that $w \notin S \cup \bigcup_{X \in \mathcal{F}} X$, we discard S without forming a feasible tree.

We remark that the static precedence rule has the same condition as the dominance rule of Fomin et al. [3] for vertex cover.

Dynamic Precedence Our dynamic precedence rule exploits the following observation:

▶ **Lemma 1.7.** *Let $\mathcal{F}$ be a collection of feasible trees. Let $v \in \bigcap_{X \in \mathcal{F}} N(X)$ be a vertex. We say that v* dynamically precedes *$\mathcal{F}$ if $N(v) \subseteq \bigcup_{X \in \mathcal{F}} X \cup N(X)$. There exists a treedepth decomposition of minimum height such that whenever a vertex $v \in V$ dynamically precedes a collection $\mathcal{F}$ of subtrees below some full separator S, then v is also contained in S.*

We utilize the dynamic precedence rule when composing feasible forests $(\mathcal{F}, h)$. For each $v \in \bigcap_{X \in \mathcal{F}} N(X)$, we check whether v dynamically precedes $\mathcal{F}$. If that is the case, v is greedily added to the full separator above $\mathcal{F}$. This rule reduces the number of feasible forests that need to be considered: in particular, supersets $\mathcal{F} \cup \{X\} \supseteq \mathcal{F}$ with $v \notin N(X)$ do not need to be constructed.

References

1 Narek Bojikian, Alexander van der Grinten, Falko Hegerfeld, Laurence Alec Kluge, and Stefan Kratsch. Pace-challenge-2020-hu-berlin-exact-track, June 2020. URL: `https://doi.org/10.5281/zenodo.3894555`, `doi:10.5281/zenodo.3894555`.

2 Fedor V. Fomin, Archontia C. Giannopoulou, and Michal Pilipczuk. Computing treedepth faster than 2^n. *Algorithmica*, 73(1):202–216, 2015. URL: `https://doi.org/10.1007/s00453-014-9914-4`, `doi:10.1007/s00453-014-9914-4`.

3 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009. URL: `https://doi.org/10.1145/1552285.1552286`, `doi:10.1145/1552285.1552286`.

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