CrossMark

Computing Tree-Depth Faster Than 2^n

**Fedor V. Fomin · Archontia C. Giannopoulou ·
Michał Pilipczuk**

Abstract A connected graph has tree-depth at most k if it is a subgraph of the closure of a rooted tree whose height is at most k. We give an algorithm which for a given n-vertex graph G, in time $\mathcal{O}^*(1.9602^n)$ computes the tree-depth of G. Our algorithm is based on combinatorial results revealing the structure of minimal rooted trees whose closures contain G.

Keywords Exact algorithms · Tree-depth · 2^n Barrier

1 Introduction

The tree-depth of a graph G, denoted $\mathbf{td}(G)$, is the minimum number k such that there is a rooted forest F, not necessarily a subgraph of G, with the following properties.

- $V(G) = V(F)$,
- Every tree in F is of height at most k, i.e. the longest path between the root of the tree and any of its leaves contains at most k vertices,
- G is a subgraph of the closure of F, which is the graph obtained from F by adding all edges between every vertex of F and the vertices contained in the path from this vertex to the root of the tree that it belongs to.

F. V. Fomin · A. C. Giannopoulou (✉) · M. Pilipczuk
Department of Informatics, University of Bergen, P.O. Box 7803, 5020 Bergen, Norway
e-mail: archontia.giannopoulou@ii.uib.no

F. V. Fomin
e-mail: fomin@ii.uib.no

M. Pilipczuk
e-mail: michal.pilipczuk@ii.uib.no

This parameter has increasingly been receiving attention since it was defined by Nešetřil and Ossona de Mendez in [13] and played a fundamental role in the theory of classes of bounded expansion [14–17]. Tree-depth is a very natural graph parameter, and due to different applications, was rediscovered several times under different names as the vertex ranking number [2], the ordered coloring [10], and the minimum height of an elimination tree of a graph [20].

From the algorithmic perspective, it has been known that the problem of computing tree-depth is NP-hard even when restricted to bipartite graphs [2,13]. However, it also admits polynomial time algorithms for specific graph classes [6,12]. For example, when the input graph is a tree its tree-depth can be computed in linear time [20]. Moreover, as tree-depth is closed under minors, the results of Robertson and Seymour [18,19] imply that the problem is in FPT when parameterized by the solution size. In [2], Bodlaender et al. showed that the computation of tree-depth is also in XP when parameterized by treewidth. From the point of view of approximation, tree-depth can be approximated in polynomial time within a factor of $\mathcal{O}(\log^2 n)$ [4], where n is the number of vertices of the input graph. Moreover, there is a simple approximation algorithm that, given a graph G, returns a forest F such that G is contained in the closure of F and the height of F is at most $2^{\mathbf{td}(G)}$ [17]. Finally, it is easy to see (and will be described in Sect. 3) that there exists an exact algorithm for the computation of tree-depth running in $\mathcal{O}^*(2^n)$ time.[1]

We are interested in tree-depth from the perspective of exact exponential time algorithms. Tree-depth is intimately related to another two well studied parameters, treewidth and pathwidth. The treewidth of a graph can be defined as the minimum taken over all possible completions into a chordal graph of the maximum clique size minus one. Similarly, path-width can be defined in terms of completion to an interval graph. One of the equivalent definitions of tree-depth is that it is the size of the largest clique in a completion to a trivially perfect graph. These graph classes form the following chain

$$\textbf{trivially perfect} \subset \textbf{interval} \subset \textbf{chordal},$$

corresponding to the parameters tree-depth, pathwidth, and treewidth.

However, while for the computation of treewidth and pathwidth there exist $\mathcal{O}^*(c^n)$, $c < 2$, time algorithms [7,8,11,21], to the best of our knowledge no such algorithm for tree-depth has been known prior to our work. In this paper, we construct the first exact algorithm which for any input graph G computes its tree-depth in time $\mathcal{O}^*(c^n)$, $c < 2$. The running time of the algorithm is $\mathcal{O}^*(1.9602^n)$. The approach is based on the structural characteristics of the minimal forest that defines the tree-depth of the graph.

The rest of the paper is organized as follows. In Sect. 2 we give some basic definitions and preliminary combinatorial results on the minimal trees for tree-depth and in Sect. 3, based on the results from Sect. 2, we present the $\mathcal{O}(1.9602^n)$ time algorithm for tree-depth. Finally, in Sect. 4 we conclude with open problems.

---

[1] The $\mathcal{O}^*(\cdot)$ notation suppresses factors that are polynomial in the input size.

2 Minimal Rooted Forests for Tree-Depth

2.1 Preliminaries

For a graph $G = (V, E)$, we use $V(G)$ to denote V and $E(G)$ to denote E. If $S \subseteq V(G)$ we denote by $G \setminus S$ the graph obtained from G after removing the vertices of S. In the case where $S = \{u\}$, we abuse notation and write $G \setminus u$ instead of $G \setminus \{u\}$. We denote by $G[S]$ the subgraph of G induced by the set S. For $S \subseteq V(G)$, the *open neighborhood* of S in G, $N_G(S)$, is the set $\{u \in G \setminus S \mid \exists v \in S : \{u, v\} \in E(G)\}$. Again, in the case where $S = \{v\}$ we abuse notation and write $N_G(v)$ instead of $N_G(\{v\})$. Given two vertices v and u we denote by $\mathbf{dist}_G(v, u)$ their distance in G. We use $\mathcal{C}(G)$ to denote the set of connected components of G.

2.2 Tree-Depth

A *rooted forest* is a disjoint union of rooted trees. The *height* of a vertex x in a rooted forest F is the number of vertices of the path from the root (of the tree to which x belongs) to x and is denoted by $\mathbf{height}(x, F)$. The height of F is the maximum height of the vertices of F and is denoted by $\mathbf{height}(F)$. Let x, y be vertices of F. The vertex x is an *ancestor* of y if x belongs to the path linking y and the root of the tree to which y belongs. The *closure* $\mathbf{clos}(F)$ of a rooted forest F is the graph with vertex set $V(F)$ and edge set $\{\{x, y\} \mid x$ is an ancestor of y in $F, x \neq y\}$. For every vertex y of F we denote by P_y the unique path linking y and the root of the tree to which y belongs, and denote by $p(y)$ the parent of y in F, i.e. the neighbor of y in P_y. Vertices whose parent is y are called the *children* of y. We call a vertex x of F a *branching point* if x is not a root of F and $\deg_F(x) > 2$ or if x is a root of F and $\deg_F(x) \geq 2$. For a vertex v of a rooted tree T, we denote by T_v the maximal subtree of T rooted in v. For example, if v is the root of T, then $T_v = T$.

Let G be a graph. The *tree-depth* of G, denoted $\mathbf{td}(G)$, is the least $k \in \mathbb{N}$ such that there exists a rooted forest F on the same vertex set as G such that $G \subseteq \mathbf{clos}(F)$ and $\mathbf{height}(F) = k$. Note that if G is connected then F must be a tree, and the tree-depth of a disconnected graph is the maximum of tree-depth among its connected components. Thus, when computing tree-depth we may focus on the case when G is connected and F is required to be a rooted tree.

With every rooted tree T of height h we associate a sequence $(t_1, t_2, t_3, \ldots)$, where $t_i = |\{v \mid \mathbf{height}(v, T) = i\}|$, $i \in \mathbb{N}$, that is, t_i is the number of vertices of the tree T of height i, $i \in \mathbb{N}$. Note that since T is finite, this sequence contains only finitely many non-zero values.

Let T_1 and T_2 be two rooted trees with heights h_1 and h_2, and corresponding sequences $(t_1^1, t_2^1, t_3^1, \ldots)$ and $(t_1^2, t_2^2, t_3^2, \ldots)$, respectively. We say that $T_1 \prec T_2$ if and only if there exists an $i \in \mathbb{N}$ such that $t_i^1 < t_i^2$ and $t_j^1 = t_j^2$, for every $j > i$. Note in particular that if $h_1 < h_2$, then taking $i = h_2$ in this definition proves that $T_1 \prec T_2$.

Definition 1 Let G be a connected graph. A rooted tree T is *minimal* for G if

1. $V(T) = V(G)$ and $G \subseteq \mathbf{clos}(T)$, and
2. There is no tree T' such that $V(T') = V(G)$, $G \subseteq \mathbf{clos}(T')$, and $T' \prec T$.

The next observation follows from the definitions of $\prec$ and of tree-depth.

Observation 1 *Let G be a connected graph and T be a rooted tree for G such that $V(T) = V(G)$, $G \subseteq \mathbf{clos}(T)$, and $\mathbf{height}(T) > \mathbf{td}(G)$. Then there exists a rooted tree T' such that $V(T') = V(G)$, $G \subseteq \mathbf{clos}(T')$, and $\mathbf{height}(T') < \mathbf{height}(T)$, and thus $T' \prec T$.*

The following combinatorial lemmata reveal the structures of minimal trees which will be handy in the algorithm.

Lemma 1 *Let T^1 be a rooted tree with root r, $v \in V(T^1)$, and T^* be a rooted tree with root r^* such that $T^* \prec T_v^1$. If T^2 is the rooted tree obtained from T^1 after considering the union of $T^1 \setminus V(T_v^1)$ with T^* and adding an edge between r^* and $p(v)$ (if $p(v)$ exists), then $T^2 \prec T^1$.*

Proof Notice first that the claim trivially holds for the case where $v = r$ as then $T_v^1 = T^1$ and $T^2 = T^*$. Thus, from now on we prove the claim assuming that $v \neq r$. Let then h be the height of the vertex $p(v)$ in T^1.

As $T^* \prec T_v^1$, there exists an index i such that the number of vertices of height i in T^* is strictly smaller than the number of vertices of height i in T_v^1, and for every $j > i$, the number of vertices of height j is equal in both T^* and T_v^1. This implies that the number of vertices of height $h + i$ in T^2 is strictly smaller than the number of vertices of height $h + i$ in T^1, and for every $j > h + i$, the number of vertices of height j is equal in both T^1 and T^2. Thus, we again conclude that $T^2 \prec T^1$. □

Lemma 2 *Let G be a connected graph. If T is a minimal tree for G with root r then for every $v \in V(T)$,*

1. *$G[V(T_v)]$ is connected,*
2. *T_v is a minimal tree for $G[V(T_v)]$, and*
3. *If $v' \in V(T_v)$ is a branching point with minimum $\mathbf{dist}_{T_v}(v, v')$ then $N_G(v) \cap V(T_u) \neq \emptyset$, for every child u of v'.*

Proof We first prove (1). Assume to the contrary that there exists a vertex $v \in V(T)$ such that the graph $G[V(T_v)]$ is not connected. Notice that we may choose v in such a way that $\mathbf{dist}_T(r, v)$ is maximum. We first exclude the case where $v = r$. Indeed, notice that if $v = r$, then $G[V(T_r)] = G$ is connected by the hypothesis. Thus, $v \neq r$. Notice also that if v is a leaf of T then T_v is the graph consisting of one vertex, so it is again connected. Therefore, v is not a leaf of T. Let $v_1, v_2, \ldots, v_p$ be the children of v. The choice of v (maximality of distance from r) implies that $G[V(T_{v_i})]$ is a connected component of $G[V(T_v)] \setminus v$, $i \in [p]$. Moreover, from the fact that $G[V(T_v)]$ is not connected, it follows that there exists at least one $i_0 \in [p]$ such that $N_G(v) \cap V(T_{v_{i_0}}) = \emptyset$. Let T' be the tree obtained from T by removing the edge $\{v, v_{i_0}\}$ and adding the edge $\{p(v), v_{i_0}\}$. Observe that $G \subseteq \mathbf{clos}(T')$. Moreover, notice that by construction of T', we may consider T' as the tree obtained from the union of $T \setminus V(T_{p(v)})$ with $T'_{p(v)}$ after adding the edge $\{p(v), p(p(v))\}$ (if $p(v) \neq r$). It is easy to see that $T'_{p(v)} \prec T_{p(v)}$. Therefore, from Lemma 1, we end up with a contradiction to the minimality of T.

To prove (2), we assume to the contrary that there exists a vertex $v \in V(T)$ such that T_v is not a minimal tree for $G[V(T_v)]$. By the hypothesis that T is a minimal tree for G, it follows that $v \neq r$. As T_v is not a minimal tree for $G[V(T_v)]$, there exists a rooted tree T' with root r' such that $V(T') = V(T_v)$, $G[V(T_v)] \subseteq \mathbf{clos}(T')$, and $T' \prec T_v$. Let now T^* be the rooted tree obtained from the union of $T \setminus V(T_v)$ with T' after adding an edge between $p(v)$ and r'. Notice then that $G \subseteq \mathbf{clos}(T^*)$. Moreover, from Lemma 1, we get that $T^* \prec T$, a contradiction to the minimality of T.

We now prove (3). Let v be a vertex of T and v' be a branching point of T_v such that $\mathbf{dist}_{T_v}(v, v')$ is minimum, that is, v' is the highest branching point in T_v. Assume to the contrary that there exists a child u of v' such that $N_G(v) \cap V(T_u) = \emptyset$. Let T' be the tree obtained from T by switching the position of the vertices v and v', where $T' = T$ if $v = v'$. Notice that $\mathbf{clos}(T) = \mathbf{clos}(T')$ and T and T' are isomorphic, hence T' is also a minimal tree for G. Moreover, children of v in T' are exactly children of v' in T. Observe also that if w is a child of v' in T, hence also a child of v in T', then $T_w = T'_w$ and $N_{G[V(T'_v)]}(V(T'_w)) \subseteq \{v\}$. As $N_G(v) \cap V(T_u) = \emptyset$, we obtain that $G[V(T'_v)]$ is not connected. However, T' is a minimal tree for G and therefore, from (1), $G[V(T'_v)]$ is connected, a contradiction. This completes the proof of the last claim and of the lemma.   □

3 Computing Tree-Depth

3.1 The Naive DP, and Pruning the Space of States

To construct our algorithm, we need an equivalent recursive definition of tree-depth.

Proposition 1 ([13]) *The* tree-depth *of a connected graph G is equal to*

$$\mathbf{td}(G) = \begin{cases} 1 & \text{if } |V(G)| = 1 \\ 1 + \min_{v \in V(G)} \max_{H \in \mathcal{C}(G \setminus v)} \mathbf{td}(H) & \text{otherwise} \end{cases} \tag{1}$$

Proposition 1 already suggests a dynamic programming algorithm computing tree-depth of a given graph G in $\mathcal{O}^*(2^n)$ time. Assume without loss of generality that G is connected, as otherwise we may compute the tree-depth of each connected component of G separately. For every $X \subseteq V(G)$ such that $G[X]$ is connected, we compute $\mathbf{td}(G[X])$ using (1). Assuming that the tree-depth of all the connected graphs induced by smaller subsets of vertices has been already computed, computation of formula (1) takes polynomial time. Hence, if we employ dynamic programming starting with the smallest sets X, we can compute $\mathbf{td}(G)$ in $\mathcal{O}^*(2^n)$ time. Let us denote this algorithm by $\mathbb{A}_0$.

The reason why $\mathbb{A}_0$ runs in pessimistic $\mathcal{O}^*(2^n)$ time is that the number of subsets of $V(G)$ inducing connected subgraphs can be as large as $\mathcal{O}(2^n)$. Therefore, if we aim at reducing the time complexity, we need to prune the space of states significantly. Let us choose some ε, $0 < \varepsilon < \frac{1}{6}$, to be determined later, and let G be a connected graph on n vertices. We define the space of states $\mathcal{S}_\varepsilon$ as follows:

$$\mathcal{S}_\varepsilon = \{S \subseteq V(G) \mid 1 \le |S| \le \left(\tfrac{1}{2} - \varepsilon\right) n \text{ and } G[S] \text{ is connected, or}$$
$$\exists X \subseteq V(G) : |X| \le \left(\tfrac{1}{2} - \varepsilon\right) n \text{ and } G[S] \in \mathcal{C}(G \setminus X)\}.$$

Observe that thus all the sets belonging to $\mathcal{S}_\varepsilon$ induce connected subgraphs of G. The subsets $S \in \mathcal{S}_\varepsilon$ considered in the first part of the definition will be called of the *first type*, and the ones considered in the second part will be called of the *second type*. Note that $V(G) \in \mathcal{S}_\varepsilon$ since it is a subset of second type for $X = \emptyset$.

Lemma 3 *If G is a graph on n vertices, then* $|\mathcal{S}_\varepsilon| = \mathcal{O}^*\left(\binom{n}{\left(\frac{1}{2} - \varepsilon\right) n}\right)$. *Moreover,* $\mathcal{S}_\varepsilon$ *may be enumerated in* $\mathcal{O}^*\left(\binom{n}{\left(\frac{1}{2} - \varepsilon\right) n}\right)$ *time.*

Proof For sets of the first type, there are at most $n \cdot \binom{n}{\left(\frac{1}{2} - \varepsilon\right) n}$ sets S of size at most $\left(\frac{1}{2} - \varepsilon\right) n$. Moreover, one can enumerate them in $\mathcal{O}^*\left(\binom{n}{\left(\frac{1}{2} - \varepsilon\right) n}\right)$ time, and for each run a polynomial-time check whether it induces a connected subgraph. For the sets of the second type, we can in the same manner enumerate all the vertex sets X of size at most $\left(\frac{1}{2} - \varepsilon\right) n$ in $\mathcal{O}^*\left(\binom{n}{\left(\frac{1}{2} - \varepsilon\right) n}\right)$ time, and for each of them take all of the at most n connected components of $G \setminus X$. $\qquad\square$

In our algorithms we store the family $\mathcal{S}_\varepsilon$ as a collection of binary vectors of length n in a prefix tree (a trie). Thus when constructing $\mathcal{S}_\varepsilon$ we can avoid enumerating duplicates, and then test belonging to $\mathcal{S}_\varepsilon$ in $O(n)$ time.

We now define the pruned dynamic programming algorithm $\mathbb{A}_\varepsilon$ that for every $X \in \mathcal{S}_\varepsilon$ computes value $\mathbf{td}_*(G[X])$ defined as follows:

$$\mathbf{td}_*(G[X]) = \begin{cases} 1 & \text{if } |X| = 1 \\ 1 + \min\limits_{v \in X} \max\limits_{H \in \mathcal{C}(G[X] \setminus v), \, V(H) \in \mathcal{S}_\varepsilon} \mathbf{td}_*(H) & \text{otherwise} \end{cases} \quad (2)$$

We use the convention that $\mathbf{td}_*(G[X]) = +\infty$ if $X \notin \mathcal{S}_\varepsilon$. The algorithm $\mathbb{A}_\varepsilon$ can be implemented in a similar manner as $\mathbb{A}$ so that its running time is $\mathcal{O}^*(|\mathcal{S}_\varepsilon|)$. We consider sets from $\mathcal{S}_\varepsilon$ in increasing order of cardinalities (sorting $|\mathcal{S}_\varepsilon|$ with respect to cardinalities takes $\mathcal{O}^*(|\mathcal{S}_\varepsilon|)$ time) and simply apply formula (2). Note that computation of formula (2) takes polynomial time, since we need to consider at most n vertices v, and for every connected component $H \in \mathcal{C}(G \setminus v)$ we can test whether its vertex set belongs to $\mathcal{S}_\varepsilon$ in $\mathcal{O}(n)$ time.

For a set $S \in \mathcal{S}_\varepsilon$ and T being a minimal tree for $G[S]$, we say that T is *covered by* $\mathcal{S}_\varepsilon$ if $V(T_v) \in \mathcal{S}_\varepsilon$ for every $v \in S$. The following lemma expresses the crucial property of $\mathbf{td}_*$.

Lemma 4 *For any connected graph G and any subset $S \subseteq V(G)$, it holds that* $\mathbf{td}_*(G[S]) \ge \mathbf{td}(G[S])$. *Moreover, if $S \in \mathcal{S}_\varepsilon$ and there exists a minimal tree T for $G[S]$ that is covered by $\mathcal{S}_\varepsilon$, then* $\mathbf{td}_*(G[S]) = \mathbf{td}(G[S])$.

Proof We first prove the first claim by induction with respect to the cardinality of S. If $\mathbf{td}_*(G[S]) = +\infty$ then the claim is trivial. Therefore, we assume that $S \in \mathcal{S}_\varepsilon$, there exists some $r \in S$ such that $\mathbf{td}_*(G[S]) = 1 + \max_{H \in \mathcal{C}(G[S] \setminus r)} \mathbf{td}_*(H)$, and $V(H) \in \mathcal{S}_\varepsilon$ for each $H \in \mathcal{C}(G[S] \setminus r)$. By the induction hypothesis, since $|V(H)| \leq |S|$ for each $H \in \mathcal{C}(G[S] \setminus r)$, we infer that $\mathbf{td}_*(H) \geq \mathbf{td}(H)$ for each $H \in \mathcal{C}(G[S] \setminus r)$. On the other hand, by (1) we have that $\mathbf{td}(G[S]) \leq 1 + \max_{H \in \mathcal{C}(G[S] \setminus r)} \mathbf{td}(H)$. Therefore,

$$\mathbf{td}(G[S]) \leq 1 + \max_{H \in \mathcal{C}(G[S] \setminus r)} \mathbf{td}(H)$$
$$\leq 1 + \max_{H \in \mathcal{C}(G[S] \setminus r)} \mathbf{td}_*(H) = \mathbf{td}_*(G[S]),$$

and the induction step follows.

We now prove the second claim, again by induction with respect to the cardinality of S. Let T be a minimal tree for $G[S]$ that is covered by $\mathcal{S}_\varepsilon$. Let r be the root of T and let $v_1, v_2, \ldots, v_p$ be the children of r in T. By (2) of Lemma 2, we have that T_{v_i} is a minimal tree for $G[V(T_{v_i})]$, for each $i \in [p]$. Moreover, since T was covered by $\mathcal{S}_\varepsilon$, then so does each T_{v_i}. By the induction hypothesis we infer that $\mathbf{td}_*(G[V(T_{v_i})]) = \mathbf{td}(G[V(T_{v_i})])$ for each $i \in [p]$, since $|V(T_{v_i})| < |S|$. Moreover, since T and each T_{v_i} are minimal, we have that

$$\mathbf{td}(G[S]) = \mathbf{height}(T) = 1 + \max_{i \in [p]} \mathbf{height}(T_{v_i}) = 1 + \max_{i \in [p]} \mathbf{td}(G[V(T_{v_i})])$$
$$= 1 + \max_{i \in [p]} \mathbf{td}_*(G[V(T_{v_i})]) \geq \mathbf{td}_*(G[S]).$$

The last inequality follows from the fact that, by (1) of Lemma 2, $G[V(T_{v_i})]$ are connected components of $G[S] \setminus r$ and moreover that their vertex sets belong to $\mathcal{S}_\varepsilon$. Hence, vertex r was considered in (2) when defining $\mathbf{td}_*(G[S])$. We infer that $\mathbf{td}(G[S]) \geq \mathbf{td}_*(G[S])$, and $\mathbf{td}(G[S]) \leq \mathbf{td}_*(G[S])$ by the first claim, so $\mathbf{td}_*(G[S]) = \mathbf{td}(G[S])$. $\square$

Lemma 4 implies that the tree-depth is already computed exactly for all connected subgraphs induced by significantly less than half of the vertices.

Corollary 1 *For any connected graph G on n vertices and any $S \in \mathcal{S}_\varepsilon$, if $|S| \leq (\frac{1}{2} - \varepsilon)n$, then $\mathbf{td}_*(G[S]) = \mathbf{td}(G[S])$.*

Proof If T is a minimal tree for $G[S]$, then for every $v \in V(T)$, $G[V(T_v)]$ is connected by (1) of Lemma 2, and $|V(T_v)| \leq (\frac{1}{2} - \varepsilon)n$. Hence, for every $v \in V(T)$ it holds that $V(T_v) \in \mathcal{S}_\varepsilon$ and the corollary follows from Lemma 4. $\square$

Finally, we observe that for any input graph G the algorithm $\mathbb{A}_\varepsilon$ already computes the tree-depth of G unless every minimal tree for G has a very special structure. Let T be a minimal tree for G. A vertex $v \in V(G)$ is called *problematic* if (i) $|V(T_v)| > (\frac{1}{2} - \varepsilon)n$, and (ii) $|V(P_{p(v)})| > (\frac{1}{2} - \varepsilon)n$. We say that a minimal tree T for G is *problematic* if it contains some problematic vertex.

Corollary 2 *For any connected graph G, if G admits a minimal tree that is not problematic, then $\mathbf{td}_*(G) = \mathbf{td}(G)$.*

Proof We prove that any minimal tree T for G that is not problematic, is in fact covered by $\mathcal{S}_\varepsilon$. Then the corollary follows from Lemma 4.

Take any $v \in V(G)$; we need to prove that $V(T_v) \in \mathcal{S}_\varepsilon$. First note that $G[V(T_v)]$ is connected by (1) of Lemma 2. Hence if $|V(T_v)| \leq \left(\frac{1}{2} - \varepsilon\right) n$, then it trivially holds that $V(T_v) \in \mathcal{S}_\varepsilon$ by the definition of $\mathcal{S}_\varepsilon$. Otherwise we have that $|V(P_{p(v)})| \leq \left(\frac{1}{2} - \varepsilon\right) n$, since v is not problematic. Note then that $N_G(V(T_v)) \subseteq V(P_{p(v)})$ and so $G[V(T_v)]$ is a connected component of $V(G) \setminus V(P_{p(v)})$. Consequently, $V(T_v)$ is a subset of second type for $X = V(P_{p(v)})$. $\qquad\square$

3.2 The Algorithm

Corollary 2 already restricts cases when the pruned dynamic program $\mathbb{A}_\varepsilon$ misses the minimal tree: this happens only when all the minimal trees for the input graph G are problematic. Therefore, it remains to investigate the structure of problematic minimal trees to find out, if some problematic minimal tree could have smaller height than the minimal tree computed by $\mathbb{A}_\varepsilon$.

Let G be the input graph on n vertices. Throughout this section we assume that G admits some problematic minimal tree T. Let v be a problematic vertex in T. Let moreover v' be the highest branching point in T_v (possibly $v' = v$ if v is already a branching point in T), or v' be the only leaf of T_v in case T_v does not contain any branching points. Let $Z = V(P_{v'})$; observe that since v is problematic, we have that $|Z| > \left(\frac{1}{2} - \varepsilon\right) n$. Let $Q_1, Q_2, \ldots, Q_a$ be all the subtrees of T rooted in $N_T(Z \setminus \{v'\})$, that is, in the children of vertices of $Z \setminus \{v'\}$ that do not belong to Z, and let $R_1, R_2, \ldots, R_b$ be the subtrees of T rooted in children of v'. Note that trees $Q_1, Q_2, \ldots, Q_a, R_1, R_2, \ldots, R_b$ are pairwise disjoint, and by the definition of a minimal tree we have that $N_G(V(Q_i)), N_G(V(R_j)) \subseteq Z$ for any $i \in [a], j \in [b]$. See Fig. 1 for reference.

Let $Q = \bigcup_{i=1}^{a} V(Q_i)$ and $R = \bigcup_{j=1}^{b} V(R_j)$. For any problematic minimal tree T and a problematic vertex v in it, we say that v *defines* the sets $Z, Q_1, Q_2, \ldots, Q_a, R_1, R_2, \ldots, R_b, Q, R$ in T.

Observation 2 *If $b > 0$, then $Z = N_G(V(R_j) \cup Q)$ for any $j \in [b]$.*

Proof As $N_G(V(Q_i)), N_G(V(R_j)) \subseteq Z$ for any $i \in [a], j \in [b]$, we have that $Z \supseteq N_G(V(R_j) \cup \bigcup_{i=1}^{a} V(Q_i))$. We proceed to proving the reverse inclusion.

Take any $z \in Z$, and let z' be the highest branching point in T_z; note that z' is always defined since $b > 0$ and thus v' is a branching point. If $z' = v'$, then by (3) of Lemma 2 we infer that $z \in N_G(V(R_j)), j \in [b]$. Otherwise, we have that $z' \in Z \setminus \{v'\}$. Since z' is a branching point, there exists some subtree Q_i rooted in a child of z'. We can again use (3) of Lemma 2 to infer that $z \in N_G(V(Q_i))$, so also $z \in N_G(Q)$. $\qquad\square$

Observe that if $b = 0$, then we trivially have that $Z = V(G) \setminus Q$.

Observation 3 $|Q| < 2\varepsilon n$.

Proof Since v is problematic, we have that $|V(P_{p(v)})| > \left(\frac{1}{2} - \varepsilon\right) n$ and $|V(T_v)| > \left(\frac{1}{2} - \varepsilon\right) n$. Observe also that $V(P_{p(v)}) \cup V(T_v) = Z \cup R$ by the definition of Z. Since $V(P_{p(v)}) \cap V(T_v) = \emptyset$ we have that:
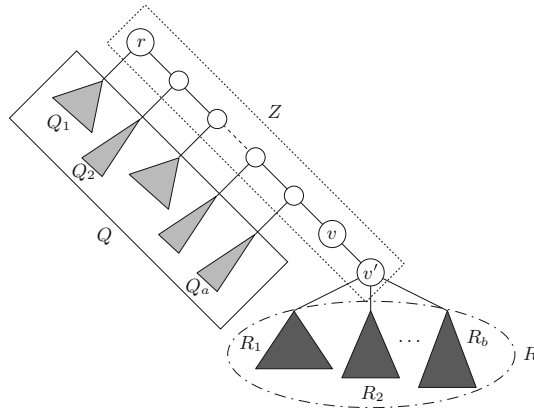
Fig. 1 A problematic minimal tree T rooted at r, the problematic vertex v, the branching vertex v' in T_v, the set Z (the set consisting of *white* vertices), and the sets Q_i, $i \in [a]$, and R_j, $j \in [b]$. We would like to remark here that the figure only aims to facilitate identification of the above sets and that it is possible that, for some such tree T, the vertices in $Z \setminus v'$ have more children or there exists a vertex q in Q such that **height**$(q, T) > $ **height**(r, T) for every vertex $r \in R$

$$|Z \cup R| = |V(P_{p(v)}) \cup V(T_v)| > (1 - 2\varepsilon)n.$$

Since $Q = V(G) \setminus (Z \cup R)$, the claim follows. $\square$

Observation 4 $|R| < \left(\frac{1}{2} + \varepsilon\right) n - |Q|.$

Proof Since $R = V(G) \setminus (Z \cup Q)$ and $|Z| > \left(\frac{1}{2} - \varepsilon\right) n$, we have that

$$|R| = |V(G) \setminus (Z \cup Q)| = n - |Z| - |Q| < \left(\frac{1}{2} + \varepsilon\right) n - |Q|.$$

$\square$

Observation 5 *If $b > 0$, then $b \geq 2$ and* $\min_{j \in [b]} |V(R_j)| < \left(\frac{1}{4} + \frac{\varepsilon}{2}\right) n - \frac{|Q|}{2}.$

Proof If $b > 0$ then v' is a branching point and it has at least two children. It follows that $b \geq 2$. For the second claim, observe that since $b \geq 2$ we have that $\min_{j \in [b]} |V(R_j)| \leq |R|/2$ and the claim follows from Observation 4. $\square$

We can proceed to the description of our main algorithm, denoted further $\mathbb{A}$. Similarly as before, without loss of generality let us assume that G is connected. First, the algorithm constructs the family $\mathcal{S}_\varepsilon$ using Lemma 3, and runs the algorithm $\mathbb{A}_\varepsilon$ on it. Note that these steps can be performed in time $\mathcal{O}^* \left(\binom{n}{(\frac{1}{2} - \varepsilon) n} \right)$. We can therefore assume that the value **td**$_*(G[S])$ is computed for every $S \in \mathcal{S}_\varepsilon$, and in particular for $S = V(G)$.

Now the algorithm proceeds to checking whether a better problematic minimal tree T with problematic vertex v can be constructed. We adopt the notation introduced

in the previous paragraphs for a problematic minimal tree T. We aim at identifying set Z and sets $V(Q_1), V(Q_2), \ldots, V(Q_a), V(R_1), V(R_2), \ldots, V(R_b)$. Without loss of generality assume that if $b > 0$, then $V(R_1)$ has the smallest cardinality among $V(R_1), V(R_2), \ldots, V(R_b)$, i.e., $|V(R_1)| \leq |V(R_2)|, \ldots, |V(R_b)|$. Let then $Y = Q \cup R_1$ if $b > 0$, and $Y = Q$ if $b = 0$.

The algorithm branches into at most $(n+1)$ subbranches, in each fixing the expected cardinality y of Y. Note that by Observations 3 and 5 and the fact that $\varepsilon < \frac{1}{6}$ we may assume that

$$y < |Q| + \left(\frac{1}{4} + \frac{\varepsilon}{2}\right) n - \frac{|Q|}{2} = \frac{|Q|}{2} + \left(\frac{1}{4} + \frac{\varepsilon}{2}\right) n < \left(\frac{1}{4} + \frac{3\varepsilon}{2}\right) n. \qquad (3)$$

Then the algorithm branches into $\binom{n}{y}$ subbranches, in each fixing a different subset of vertices of size smaller than y as the set Y. Note that sets $V(Q_1)$, $V(Q_2), \ldots, V(Q_a), V(R_1)$ are then defined as vertex sets of connected components of $G[Y]$. The algorithm branches further into $(n + 1)$ cases. In one case the algorithm assumes that $b = 0$ and therefore concludes that $Q = Y$. In other cases the algorithm assumes that $b > 0$ and picks one of the components of $G[Y]$ assuming that its vertex set is $V(R_1)$, thus recognizing Q as $Y \setminus V(R_1)$, i.e., the union of vertex sets of remaining components of $G[Y]$.

In the case when $b = 0$ the algorithm concludes that $Z = V(G) \setminus Q$. In the cases when $b > 0$, the algorithm concludes that $Z = N_G(Y)$ using Observation 2. Having identified Z, the sets $V(R_1), V(R_2), \ldots, V(R_j)$ can be recognized as vertex sets of connected components of $V(G) \setminus (Z \cup Q)$. Observations 2, 3, and 5 ensure that for every problematic minimal tree T for G, there will be at least one subbranch where sets $Z, V(Q_1), V(Q_2), \ldots, V(Q_a), V(R_1), V(R_2), \ldots, V(R_b)$ are fixed correctly. Observe also that in each of at most $(n + 1)$ branches where y has been fixed, we produced at most $(n + 1) \cdot \binom{n}{y}$ subbranches. We perform also sanity checks: whenever any produced branch does not satisfy any of Observations 2, 3, 4 or 5, or the fact that $V(R_1)$ is a smallest set among $V(R_1), V(R_2), \ldots, V(R_j)$, we terminate the branch immediately.

The algorithm now computes $\mathbf{td}(G[V(Q_i)])$ and $\mathbf{td}(G[V(R_j)])$ for all $i \in [a]$, $j \in [b]$. Recall that by Corollary 1, for any set $X \subseteq V(G)$ such that $G[X]$ is connected and $|X| \leq (\frac{1}{2} - \varepsilon)n$, we have that $\mathbf{td}(G[X]) = \mathbf{td}_*(G[X])$, and hence the value $\mathbf{td}(G[X])$ has been already computed by algorithm $\mathbb{A}_\varepsilon$. Since $|Q| \leq 2\varepsilon n$ and $\varepsilon < \frac{1}{6}$, we infer that this is the case for every set $V(Q_i)$ for $i \in [a]$, and values $\mathbf{td}(G[V(Q_i)])$ are already computed. The same holds for every R_j assuming that $|V(R_j)| \leq (\frac{1}{2} - \varepsilon)n$.

Assume then that there exists some j_0 such that $|V(R_{j_0})| > (\frac{1}{2} - \varepsilon)n$, i.e., we have no guarantee that the algorithm $\mathbb{A}_\varepsilon$ computed $\mathbf{td}(G[V(R_{j_0})])$ correctly. Note that by Observation 4 and the fact that $\varepsilon < \frac{1}{6}$, there can be at most one such j_0. Furthermore, if this is the case, then by Observation 5 we have that $b \geq 2$ and $V(R_{j_0})$ cannot be the smallest among sets $V(R_1), V(R_2), \ldots, V(R_b)$; hence, $j_0 \neq 1$ and $V(R_{j_0}) \subseteq V(G) \setminus (Z \cup Y)$. Therefore, we must necessarily have that

$$y = |Y| \leq |V(G)| - |Z| - |V(R_{j_0})| < n - \left(\frac{1}{2} - \varepsilon\right) n - \left(\frac{1}{2} - \varepsilon\right) n = 2\varepsilon n,$$

and moreover

$$|V(R_{j_0})| \leq |V(G)| - |Z| - |Y| < n - \left(\frac{1}{2} - \varepsilon\right)n - y = \left(\frac{1}{2} + \varepsilon\right)n - y.$$

Formally, if none of these assertions holds, the branch would be terminated by the sanity check. To compute $\mathbf{td}(G[V(R_{j_0})])$ we employ the naive dynamic programming routine on $G[V(R_{j_0})]$, i.e., algorithm $\mathbb{A}_0$. Observe, however, that all subsets of R_{j_0} of size at most $(\frac{1}{2} - \varepsilon)n$ belong to $\mathcal{S}_\varepsilon$. Thus from Corollary 1 their values have already been computed accurately by the algorithm $\mathbb{A}_\varepsilon$. Therefore, in this application we do not need to recompute the values for subsets of size at most $(\frac{1}{2} - \varepsilon)n$ but only the values for the subsets of R_{j_0} of greater size. Hence, we need to compute the values for at most $\sum_{i=1}^{2\varepsilon n - y} \binom{(\frac{1}{2} + \varepsilon)n - y}{(\frac{1}{2} - \varepsilon)n + i}$ subsets of R_{j_0}. As $|R_{j_0}| \leq \left(\frac{1}{2} + \varepsilon\right)n - y$ and $\varepsilon < \frac{1}{6}$, the application of algorithm $\mathbb{A}$ takes at most $\mathcal{O}^*\left(\binom{(\frac{1}{2} + \varepsilon)n - y}{(\frac{1}{2} - \varepsilon)n}\right)$ time.

Summarizing, for every choice of y (recall that, from Eq. 3, $y < \left(\frac{1}{4} + \frac{3\varepsilon}{2}\right)n$), the algorithm produced at most $(n+1) \cdot \binom{n}{y}$ branches, and in branches with $y < 2\varepsilon n$ it could have used extra $\mathcal{O}^*\left(\binom{(\frac{1}{2} + \varepsilon)n - y}{(\frac{1}{2} - \varepsilon)n}\right)$ time for computing values $\mathbf{td}(G[V(R_j)])$ whenever there was no guarantee that algorithm $\mathbb{A}_\varepsilon$ computed them correctly.

We arrive at the situation where in each branch the algorithm already identified set Z, sets $V(Q_1), V(Q_2), \ldots, V(Q_a), V(R_1), V(R_2), \ldots, V(R_b)$, and values $\mathbf{td}(G[V(Q_i)])$ and $\mathbf{td}(G[V(R_j)])$ for $i \in [a]$, $j \in [b]$. Note, however, that the algorithm does not have yet the full knowledge of the shape of tree T, because we have not yet determined in which order the vertices of Z appear on the path $P_{v'}$, and thus we do not know where the trees Q_i and R_j are attached to this path. Fortunately, it turns out that finding an optimum such ordering of vertices of Z is polynomial-time solvable.

For $i \in [a + b]$ let $M_i = Q_i$ if $i \leq a$ and $M_i = R_{i-a}$ otherwise, and let $h_i = \mathbf{td}(G[V(M_i)])$. Note that since T is minimal, by (2) of Lemma 2 we have that $h_i = \mathbf{height}(M_i)$ for each $i \in [a + b]$. Let also $Z_i = N_G(V(M_i))$; note that since $G \subseteq \mathbf{clos}(T)$, we have that $Z_i \subseteq Z$. Let σ be an ordering of Z, i.e., σ is a bijective function from Z to $[|Z|]$. Finally, we define the *weight* of σ as follows:

$$\mu(\sigma) = \max\left(|Z|, \max_{i \in [a+b]} (\max(\sigma(Z_i)) + h_i)\right). \tag{4}$$

Lemma 5 *Let G be the input graph, and let Z, $\{V(M_i)\}_{i \in [a+b]}$ be any partitioning of vertices of G such that $Z_i = N_G(V(M_i))$ is a subset of Z for any $i \in [a + b]$. Moreover, let $h_i = \mathbf{td}(G[V(M_i)])$ and for σ being an ordering of Z, let $\mu(\sigma)$ be defined by (4). Then $\mathbf{td}(G) \leq \mu(\sigma)$ for any ordering σ of Z. However, if G admits a problematic minimal tree T and Z, $\{V(M_i)\}_{i \in [a+b]}$ are defined by any problematic vertex in this tree, then $\mathbf{td}(G) = \min_\sigma \mu(\sigma)$.*

Proof We first prove that $\mathbf{td}(G) \leq \mu(\sigma)$ for any such partitioning $\{V(M_i)\}_{i\in[a+b]}$, Z of $V(G)$ and ordering σ of Z. Construct a rooted tree T' as follows. First, create a path on vertex set Z, where the vertices are ordered as in ordering σ and $\sigma^{-1}(1)$ is the root of the tree. Then, for every $i \in [a + b]$ construct a minimal tree T_i for $G[V(M_i)]$, and attach its root using one edge to the vertex $\sigma^{-1}(m_i)$, where $m_i = \max(\sigma(Z_i))$. Observe that every neighbor of $V(M_i)$ is before $\sigma^{-1}(m_i)$ in the ordering σ, and hence it follows that $G \subseteq \mathbf{clos}(T')$. Consequently, $\mathbf{td}(G) \leq \mathbf{height}(T')$. However, by the definition of T' and of $\mu(\sigma)$, we have $\mathbf{height}(T') = \mu(\sigma)$. Thus $\mathbf{td}(G) \leq \mu(\sigma)$.

We proceed to the second claim. Assume that T is a problematic minimal tree for G and assume that $Z, \{V(M_i)\}_{i\in[a+b]}$ are defined by any problematic vertex v in this tree. We adopt the notation used for T in this section. Let σ_0 be the order of vertices of Z on the path $P_{v'}$. For a tree M_i, for $i \in [a + b]$, let $z_i \in Z$ be the parent of the root of M_i; hence, for $i > a$ we have $z_i = v'$. Observe that, then $\mathbf{td}(G) = \mathbf{height}(T) = \max(|Z|, \max_{i\in[a+b]} \sigma_0(z_i) + h_i)$. Since $G \subseteq \mathbf{clos}(T)$, we infer that $\sigma_0(z_i) \geq \sigma_0(w)$ for any $w \in Z_i$. Hence $\mathbf{height}(T) \geq \mu(\sigma_0)$ by the definition of μ. Consequently, $\mathbf{td}(G) \geq \mu(\sigma_0)$, and so $\mathbf{td}(G) = \min_\sigma \mu(\sigma)$ by the first claim. □

We are left with the following scheduling problem. Given a set Z of size at most n, a family number of subsets $Z_i \subseteq Z$ for $i \in [a + b]$ and corresponding integers $h_i \leq n$, we would like to compute the minimum possible $\mu(\sigma)$ among orderings σ of Z. Let this problem be called minimum ordering with independent delays (MOID, for short).

Lemma 6 *Minimum ordering with independent delays is polynomial-time solvable.*

Proof Observe that since $|Z| \leq n$ and $h_i \leq n$, for any ordering σ we have that $\mu(\sigma) \leq 2n$. We therefore iterate through all possible values M from $|Z|$ to $2n$, and for each M we check whether there exists some σ with $\mu(\sigma) \leq M$. The first M for which this test returns a positive outcome is equal to $\min_\sigma \mu(\sigma)$.

For a given M, construct an auxiliary bipartite graph H with Z on one side and $\{1, 2, \ldots, |Z|\}$ on the other side. We put an edge between an element z and an index j if and only if the following holds: for every Z_i to which z belongs, it holds that $j + h_i \leq M$. It is easy to verify that orderings σ of Z with $\mu(\sigma) \leq M$ correspond one-to-one to perfect matchings in H. Indeed, if we are given an ordering σ with $\mu(\sigma) \leq M$, then we have that for every $z \in Z$ and Z_i to which z belongs, it holds that $\sigma(z) + h_i \leq M$ by the definition of $\mu(\sigma)$. Hence, $\{z, \sigma(z)\}$ is an edge in H and $\{\{z, \sigma(z)\} \mid z \in Z\}$ is a perfect matching in H. On the other hand, if we are given a perfect matching $\{\{z, j_z\} \mid z \in Z\}$ in H, then we may define an ordering σ of Z by putting $\sigma(z) = j_z$. Then for every $z \in Z$ and Z_i to which z belongs, we have that $\{z, \sigma(z)\}$ is an edge in H and, consequently, $\sigma(z) + h_i \leq M$. As we chose z and Z_i arbitrarily, it follows that $\max_{i\in[a+b]} (\max(\sigma(Z_i)) + h_i) \leq M$ and so $\mu(\sigma) \leq M$.

Therefore, to solve the MOID problem it suffices to construct H in polynomial time and run any polynomial-time algorithm for finding a perfect matching in H.   □

We remark that MOIDs can be also solved in $\mathcal{O}(n + \sum_{i=1}^{a+b} |Z_i|)$ time using greedy arguments. Since we are not interested in optimizing polynomial factors, in the proof of Lemma 6 we used the more concise matching argument to keep the description

simple. We leave finding a faster algorithm for MOID to the reader as an interesting exercise.

Concluding, in every subbranch algorithm $\mathbb{A}$ constructs an instance of MOID and solves it in polynomial time using the algorithm of Lemma 6. Lemma 5 ensures that none of the values found in subbranches will be larger than $\mathbf{td}(G)$, and that if G admits a problematic minimal tree T then $\mathbf{td}(G)$ will be found in at least one subbranch. Therefore, by Corollary 2 we can conclude the algorithm $\mathbb{A}$ by outputting the minimum of $\mathbf{td}_*(G)$, computed by $\mathbb{A}_\varepsilon$, and the values returned by subbranches.

Algorithm 1 Computing the tree-depth of a connected graph

Enumerate $\mathcal{S}_\varepsilon$.
Run algorithm $\mathbb{A}_\varepsilon$ on $\mathcal{S}_\varepsilon$.
for all $i \leq (\frac{1}{4} + \frac{3\varepsilon}{2})n$ **do**
  for all $Y \subseteq V(G)$ with $y = |Y| = i$ **do**
    for all $R_1 \subseteq Y$ with $R_1 = \emptyset$ or R_1 induces a connected component of $G[Y]$ **do**
      if $R_1 \neq \emptyset$ **then**
      $Z \leftarrow N_G(Y)$
      for all connected components C of $V(G) \setminus (Z \cup Y)$ with $|C| > (\frac{1}{2} - \varepsilon)n$
      do
        Compute $\mathbf{td}(C)$ by running $\mathbb{A}_0$ on the subsets of $V(C)$ that have size greater than $(\frac{1}{2} - \varepsilon)n$.
      end for
      else
      $Z \leftarrow V(G) \setminus Q$
      end if
      Run the algorithm for the MOID problem to compute the correct order of the vertices in Z and
      subsequently $\mathbf{td}(G)$.
    end for
  end for
end for

Let us proceed with the analysis of the running time of algorithm $\mathbb{A}$. First, we have enumerated $\mathcal{S}_\varepsilon$ and run the algorithm $\mathbb{A}_\varepsilon$, which took

$$T_1(n) = \mathcal{O}^*\left(\binom{n}{(\frac{1}{2} - \varepsilon)n}\right)$$

time. Then we created a number of subbranches. For every subbranch with $y \geq 2\varepsilon n$, we have spent polynomial time (the algorithm will not enter the fourth for loop in Algorithm 1), and the number of these subbranches is bounded by $(n+1)^2 \cdot \binom{n}{(\frac{1}{4} + \frac{3\varepsilon}{2})n}$ since $y < \left(\frac{1}{4} + \frac{3\varepsilon}{2}\right)n$ and $\varepsilon < \frac{1}{6}$. Hence, on these subbranches we spent

$$T_2(n) = \mathcal{O}^*\left(\binom{n}{(\frac{1}{4} + \frac{3\varepsilon}{2})n}\right)$$

time in total. Finally, for every subbranch with $y < 2\varepsilon n$, as the fourth for loop will be entered at most once, we have spent at most $\mathcal{O}^*\left(\binom{(\frac{1}{2} + \varepsilon)n - y}{(\frac{1}{2} - \varepsilon)n}\right)$ time. As the number of such branches is bounded by $(n+1) \cdot \binom{n}{y}$, the total time spent on these branches is

$$T_3(n) = \mathcal{O}^*\left(\max_{y<2\varepsilon n}\left(\binom{n}{y}\cdot\binom{(\frac{1}{2}+\varepsilon)n-y}{(\frac{1}{2}-\varepsilon)n}\right)\right).$$

If we now let $\varepsilon = \frac{1}{10}$, then $T_1(n), T_2(n) = \mathcal{O}^*\left(\binom{n}{\frac{2}{5}n}\right) = \mathcal{O}^*(1.9602^n)$. It can be also easily shown that for any $y < \frac{1}{5}n$, it holds that $\binom{n}{y}\cdot\binom{\frac{3}{5}n-y}{\frac{2}{5}n} = \mathcal{O}^*(1.9602^n)$. To prove this, we can use the following simple combinatorial bound: $\binom{n_1}{k_1}\cdot\binom{n_2}{k_2} \le \binom{n_1+n_2}{k_1+k_2}$. This inequality can be proved by combinatorial interpretation as follows: every choice of k_1 elements from a set of size n_1 and of k_2 elements from a set of size n_2, defines uniquely a choice of $k_1 + k_2$ elements from the union of these sets, which is of size $n_1 + n_2$. Therefore, we obtain:

$$\binom{n}{y}\cdot\binom{\frac{3}{5}n-y}{\frac{2}{5}n} = \binom{n}{y}\cdot\binom{\frac{3}{5}n-y}{\frac{1}{5}n-y} \le \binom{\frac{8}{5}n-y}{\frac{1}{5}n} \le \binom{\frac{8}{5}n}{\frac{1}{5}n} = \mathcal{O}^*(1.828^n).$$

Consequently, $T_1(n), T_2(n), T_3(n) = \mathcal{O}^*(1.9602^n)$, and the whole algorithm runs in $\mathcal{O}^*(1.9602^n)$ time.

4 Conclusion

In this work we gave the first exact algorithm computing the tree-depth of a graph faster than $\mathcal{O}^*(2^n)$. As Bodlaender et al. [3] observe, both pathwidth and treewidth can be reformulated as vertex ordering problems and thus computed by a simple dynamic programming algorithm similar to the classical Held–Karp algorithm in time $\mathcal{O}^*(2^n)$ [9]. For example, computing the optimum value of treewidth is equivalent to finding an elimination ordering which minimizes the sizes of cliques created during the elimination process. As far as tree-depth is concerned, Nešetřil and Ossona de Mendez [17] give an alternative definition of tree-depth in terms of *weak-colorings*, which in turn are defined also via vertex orderings; however, it is unclear whether this definition can be used for an algorithm working in $\mathcal{O}^*(2^n)$ time. Interestingly enough, for many of vertex ordering problems, like Hamiltonicity, treewidth, or pathwidth, an explicit algorithm working in time $\mathcal{O}^*(c^n)$ for some $c < 2$ can be designed, see [1,7, 21]. On the other hand, for several other vertex ordering problems no such algorithms are known. The two natural problems to attack are (i) the computation of cutwidth, and (ii) the Minimum Feedback Arc Set in Digraph problem; see [3,5] for definitions and details. It is known that the cutwidth of a graph can be computed in time $\mathcal{O}^*(2^t)$, where t is the size of a vertex cover in the graph [5]; thus the problem is solvable in time $\mathcal{O}^*(2^{n/2})$ on bipartite graphs. We leave existence of faster exponential algorithms for these problems as an open question.

References

1. Björklund, A.: Determinant sums for undirected hamiltonicity. In: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010), IEEE, pp. 173–182 (2010)
2. Bodlaender, H.L., Deogun, J.S., Jansen, K., Kloks, T., Kratsch, D., Müller, H., Tuza, Z.: Rankings of graphs. SIAM J. Discret. Math. **11**(1), 168–181 (1998)
3. Bodlaender, H.L., Fomin, F.V., Koster, A.M.C.A., Kratsch, D., Thilikos, D.M.: A note on exact algorithms for vertex ordering problems on graphs. Theory Comput. Syst. **50**(3), 420–432 (2012)
4. Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J. Algorithms **18**(2), 238–255 (1995)
5. Cygan, M., Lokshtanov, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: On cutwidth parameterized by vertex cover. Algorithmica. 1–14 (2012)
6. Deogun, J.S., Kloks, T., Kratsch, D., Müller, H.: On the vertex ranking problem for trapezoid, circular-arc and other graphs. Discret. Appl. Math. **98**(1–2), 39–63 (1999)
7. Fomin, F.V., Kratsch, D., Todinca, I., Villanger, Y.: Exact algorithms for treewidth and minimum fill-in. SIAM J. Comput. **38**(3), 1058–1079 (2008)
8. Fomin, F.V., Villanger, Y.: Treewidth computation and extremal combinatorics. Combinatorica **32**(3), 289–308 (2012)
9. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. J. SIAM **10**, 196–210 (1962)
10. Katchalski, M., McCuaig, W., Seager, S.M.: Ordered colourings. Discret. Math. **142**(1–3), 141–154 (1995)
11. Kitsunai, K., Kobayashi, Y., Komuro, K., Tamaki, H., Tano, T.: Computing directed pathwidth in $O(1.89^n)$ time. In: Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC). Lecture Notes in Computer Science, vol. 7535, pp. 182–193. Springer (2012)
12. Kloks, T., Müller, H., Wong, C.K.: Vertex ranking of asteroidal triple-free graphs. Inf. Process. Lett. **68**(4), 201–206 (1998)
13. Nešetřil, J., de Mendez, P.O.: Tree-depth, subgraph coloring and homomorphism bounds. Eur. J. Comb. **27**(6), 1022–1041 (2006)
14. Nešetřil, J., de Mendez, P.O.: Grad and classes with bounded expansion I. Decompositions. Eur. J. Comb. **29**(3), 760–776 (2008)
15. Nešetřil, J., de Mendez, P.O.: Grad and classes with bounded expansion II. Algorithmic aspects. Eur. J. Comb. **29**(3), 777–791 (2008)
16. Nešetřil, J., de Mendez, P.O.: Grad and classes with bounded expansion III. Restricted graph homomorphism dualities. Eur. J. Comb. **29**(4), 1012–1024 (2008)
17. Nešetřil, J., de Mendez, P. O.: Sparsity—Graphs, Structures, and Algorithms, volume 28 of Algorithms and combinatorics. Springer (2012)
18. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Comb. Theory, Ser. B **63**(1), 65–110 (1995)
19. Robertson, N., Seymour, P.D.: Graph Minors. XX. Wagner's conjecture. J. Comb. Theory, Ser. B **92**(2), 325–357 (2004)
20. Schäffer, A.A.: Optimal node ranking of trees in linear time. Inf. Process. Lett. **33**(2), 91–96 (1989)
21. Suchan, K., Villanger, Y.: Computing pathwidth faster than 2^n. In: Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC 2009), vol. 5917, pp. 324–335. Springer (2009)