# MaxSAT-Based Postprocessing for Treedepth

Vaidyanathan Peruvemba Ramaswamy(✉) and Stefan Szeider

Algorithms and Complexity Group, TU Wien, Vienna, Austria
{vaidyanathan,sz}@ac.tuwien.ac.at

**Abstract.** Treedepth is an increasingly popular graph invariant. Many NP-hard combinatorial problems can be solved efficiently on graphs of bounded treedepth. Since the exact computation of treedepth is itself NP-hard, recent research has focused on the development of heuristics that compute good upper bounds on the treedepth.

In this paper, we introduce a novel MaxSAT-based approach for improving a heuristically obtained treedepth decomposition. At the core of our approach is an efficient MaxSAT encoding of a weighted generalization of treedepth arising naturally due to subtree contractions. The encoding is applied locally to the given treedepth decomposition to reduce its depth, in conjunction with the collapsing of subtrees. We show the local improvement method's correctness and provide an extensive experimental evaluation with some encouraging results.

**Keywords:** Tree-depth · Elimination-tree height · SAT encoding · MaxSAT · Computational experiments

## 1 Introduction

The treedepth [29,30] of a connected graph $G$ is the smallest integer $k$ such that $G$ is a subgraph of the transitive closure $[T]$ of a tree $T$ of height $k$. The transitive closure $[T]$ is obtained from $T$ by adding all edges $uv$ whenever $u$ is an ancestor of $v$ in $T$ (see Fig. 1 for an example). We call $T$ a *treedepth decomposition* of $G$ of depth $k$. The notion of treedepth was first investigated employing *elimination trees* (*e-trees*) and *elimination height* [18,20,22,34]. 1-*partition trees* [16] and *separation game* [23] are some other names used in literature for alternative approaches to treedepth.

Treedepth has algorithmic applications for several problems where treewidth cannot be used [8,10,15]. It admits algorithms for these problems whose running times are exponential in the treedepth but polynomial (of constant order) in the input size. These results request methods for computing treedepth decompositions of small (ideally minimal) depth, which is generally an NP-hard task [36].

The authors acknowledge the support by the FWF (projects P32441 and W1255) and by the WWTF (project ICT19-065).


H. Simonis (Ed.): CP 2020, LNCS 12333, pp. 478–495, 2020.
https://doi.org/10.1007/978-3-030-58475-7_28

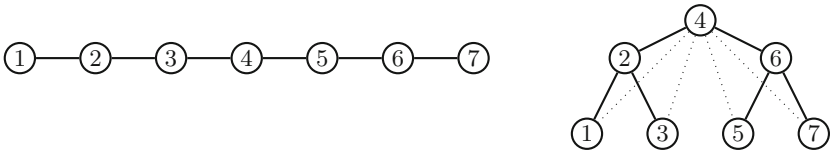



**Fig. 1.** Left: graph $P_7$. Right: treedepth decomposition of $P_7$ of depth 3.

Exact algorithms for computing the treedepth of graphs have been suggested in theoretical work [6,38]. Until recently, only very few practical implementations of algorithms that compute treedepth decompositions have been reported in the literature. Villaamil [42] discussed several heuristic methods based on minimal separators, and Ganian et al. [9] suggested two exact methods based on SAT-encodings. In general, exact methods are limited to small graphs (up to around 50 vertices [9]), whereas heuristic methods apply to large graphs but can get stuck at suboptimal solutions.

*Contribution.* In this paper, we propose the novel MaxSAT-based algorithm TD-SLIM that provides a crossover between exact and heuristic methods, taking the best of two worlds. The basic idea is to take a solution computed by a heuristic method and apply an exact (MaxSAT-based) method locally to parts of the solution, chosen small enough to admit a feasible encoding size. Although the basic idea sounds compelling and reasonably simple, its realization requires several conceptual contributions and novel results.

At the heart of our approach, local parts of the treedepth decomposition must reflect certain properties of the global decomposition. That way, an improved local decomposition can be fit back into the global one. This additional information gives rise to the more general decomposition problem, namely the *treedepth decomposition of weighted graphs with ancestry constraints*, for which we present a partition-based characterization, which leads to an efficient MaxSAT encoding. As the weights can become large, a distinctive feature of our encoding is that its size remains independent of the weights appearing in the instance.

We establish the correctness of our local-improvement method and provide an experimental evaluation on various benchmark graphs. Thereby, we compare different parameter settings and configurations and the effect of replacing several SAT calls by one MaxSAT call.

Our findings are significant, as they show that an improvement in the initial decomposition is feasible in practically all cases. The best configuration could, on average, almost reduce the depth of the initial treedepth decomposition by a half (52%) for a simple heuristic (DFS) and by a third (29%) when starting from a more elaborate heuristic (Sep). Somewhat surprisingly, it turned out that on smaller instances, that admit a single SAT encoding, the SAT-based local improvement method outperforms a single SAT call, achieving a similar depth in a tenth of the time.

## 2 Related Work

The idea of using SAT encodings for computing a decompositional graph invariant is due to Samer and Veith [39]. They proposed an *ordering-based* encoding for treewidth, which provides the basis for several subsequent improved encodings [1,2]. Heule and Szeider [13] proposed a SAT encoding for clique-width, thereby introducing a first *partition-based* encoding. These two general approaches (ordering-based and partition-based encodings) have been worked out and compared for several other decompositional graph invariants (also known as width parameters) [9,25]. Recently, several papers have proposed SMT-encodings for decompositional hypergraph parameters [4,40].

All the encodings mentioned above suffer from the limitation that the encoding size is at least cubic in the size of the input graph or hypergraph, which limits the practical applicability of these methods to graphs or hypergraph whose size is in the order of several hundred vertices. Lodha et al. [24,26] introduced the SAT-based local improvement method (SLIM), which extends the applicability of SAT-encodings to larger inputs. So far, there have been three concrete approaches that use SLIM, one for computing branchwidth (in the papers cited above [24,26]), one for computing treewidth [5], and very recently one for treewidth-bounded Bayesian network structure learning [33]. SLIM is a meta-heuristic that, similarly to Large Neighborhood Search [35], tries to improve a current solution by exploring its neighborhood of potentially better solutions. As a distinctive feature, SLIM defines neighboring solutions in a structurally highly constrained way and uses a complete method (SAT) to identify a better solution.

The comprehensive thesis by Villaamil [42] discusses four heuristics for computing treedepth. The first heuristic is based on computing a depth-first search (DFS) spanning tree of the given graph, which happens to be a valid treedepth decomposition. The remaining three heuristics are all based on finding minimal separators. Two of them are greedy local-search techniques while the third one makes use of a spectral algorithm to compute the separators, providing better decompositions at the expense of longer running times [37]. Several algorithms have been proposed for minimizing the height (among other metrics) of e-trees in the context of matrix factorizations [11,19,21,28]. More specifically, treedepth has been studied in the area of CSP under the name pseudo-tree height [3,7,17,19]. However, only few papers focus on minimizing the treedepth alone, they usually minimize a secondary measure such as fill-in. Very recently, due to the PACE Challenge 2020[1], the implementations of several new heuristics for computing treedepth decompositions became available.

## 3 Preliminaries

All considered graphs are finite, simple, and undirected. Let $G$ be a graph. $V(G)$ and $E(G)$ denote the vertex set and the edge set of $G$, respectively. The size

[1] https://pacechallenge.org/2020.

of the graph, denoted by $|G|$, is the number of vertices, i.e., $|V(G)|$. We denote an edge between vertices $u$ and $v$ by $uv$ (or equivalently by $vu$). The subgraph of $G$ induced by a set $S \subseteq V(G)$, denoted by $G[S]$, has as vertex set $S$ and as edge set $\{\, uv \in E(G) \mid u, v \in S \,\}$. As a shorthand, we sometimes use $G[H]$ to represent $G[V(H)]$, where $G, H$ are graphs. For a set $S \subseteq V(G)$ we define $G - S = G[V(G) \setminus S]$.

For a rooted tree $T$ and a vertex $v \in V(T)$, we let $T_v$ denote the *subtree of $T$ rooted at $v$*. The vertex $v$ is the *parent* of vertex $u$ if $v$ is the first vertex (after $u$) on the path from $u$ to the root of the tree. A vertex $v$ is an *ancestor* of vertex $u$ if $v$ lies on the path from $u$ to the root of the tree and $v \neq u$. We use $r(T)$ to denote the root of a tree $T$. The *height of a vertex $v$* in a rooted forest $T$, denoted by $\mathit{height}_T(v)$, is 1 plus the length of a longest path from $v$ to any leaf not passing through any ancestor (the height of a leaf is 1). The *height of a rooted tree $T$* is then $\mathit{height}_T(r(T))$, i.e., the height of the root. Naturally, the height of a rooted forest is the maximum height of its constituent trees. The *depth of a vertex $v$* in a tree $T$, denoted by $\mathit{depth}_T(v)$, is the length of the path from $r(T)$ to $v$ (the depth of $r(T)$ is 1). The *transitive closure* $[T]$ of a rooted forest $T$ is the undirected graph with vertex set $V(T)$ having an edge between two vertices if and only if one is an ancestor of the other in $T$.

The *treedepth* of an undirected graph $G$, denoted by $\mathit{td}(G)$, is the smallest integer $k$ such that there is a rooted forest $T$ with vertex set $V(G)$ of height $k$ for which $G$ is a subgraph of $[T]$. A forest $T$ for which $G$ is a subgraph of $[T]$ is also called a *treedepth decomposition*, whose *depth* is equal to the height of the forest. Informally, a graph has treedepth at most $k$ if it can be embedded in the closure of a forest of height $k$. If $G$ is connected, then it can be embedded in the closure of a tree instead of a forest.

Alternatively, letting $C(G)$ denote the set of connected components of a graph $G$, the treedepth of $G$ can be defined recursively as follows [29]:

$$\mathit{td}(G) = \begin{cases} 1 & \text{if } |V(G)| = 1; \\ \max_{G' \in C(G)} \mathit{td}(G') & \text{if } |C(G)| > 1; \\ 1 + \min_{v \in V(G)} \mathit{td}(G - v) & \text{otherwise.} \end{cases}$$

Based on this definition, it is clear that when we want to compute a treedepth decomposition $T$ of a graph $G$, we can assume without loss of generality, that $G$ is connected, and that the decomposition $T$ is a rooted tree.

## 4 Local Improvement

In this section, we lay out the theoretical foundations of our approach. Due to space constraints, we have omitted some proofs. For this section, let us fix a connected graph $G$ for which we would like to obtain a treedepth decomposition. We assume that $G$ is too large to admit a practically feasible SAT encoding. Hence, we can use a heuristic method (i.e., a *global solver*) to obtain a possibly suboptimal treedepth decomposition $T$. Now we would like to use a SAT-based

encoding (i.e., a *local solver*) to possibly improve the decomposition $T$ and reduce its depth. We refer to this approach as the *SAT-based Local Improvement Method for Treedepth* (TD-SLIM).

In our description of the procedure, we use a *stack* data-structure, which is essentially an ordered set allowing constant-time access, insertion, and deletion at the last position. We denote an empty stack by $\emptyset$. Inserting a new element at the last position is called *pushing*. Retrieving and deleting the last element is called *popping*. Finally, we denote the last element of a stack $S$ by $last(S)$.

A first idea is to select subtrees $T_v$ at nodes $v$ of $T$ that are located low enough in $T$ such that $G[T_v]$ is small enough to admit a SAT encoding but $G[T_u]$ is not, where $u$ is the parent of $v$. This way, we could try all such $v$ and successively replace subtrees $T_v$ with new subtrees $T'_v$ of possibly lower height as long as $[T'_v]$ contains all the edges of the induced graph $G[T_v]$. All other edges of $G$, in particular those with one end in $T_v$ and one end outside $T_v$, still remain included in the overall closure. This simple procedure, however, can only improve those parts of $T$ that are close to the leaves, therefore, leaving large parts of $T$ untouched.

In order to overcome this limitation, we use an operation that contracts an entire subtree $T_v$ of $T$ into the single node $v$, and we label $v$ with the depth of $T_v$. With such contractions, we can successively eliminate lower parts of $T$, so that eventually all parts of $T$ become accessible to a potential local improvement. This requires the local solver not only to deal with weighted vertices, but now, the edges outside the induced graph $G[T_v]$ are "not safe" anymore, and one must take special care for that. We accomplish this by labeling $v$ with a set $A(v)$ of *ancestors*, which are all the vertices $u \in V(G) \setminus V(T_v)$, which are adjacent in $G$ with a vertex $w \in V(T_v) \setminus \{v\}$. Since $T$ is a treedepth decomposition, all ancestors of $v$ lie on the path between the root of $T$ and $v$. We add two more labels to $v$: $d(v)$ holding the weighted depth of $T_v$ (which we formally define in the next subsection) and $S(v)$ which is a stack maintaining the sequence of subtrees rooted at $v$ that were contracted, i.e., after contracting $T_v$, we push $T_v$ onto $S(v)$ so that we can reverse the contraction later. We refer to the process of storing or associating a decomposition with a particular vertex as "tagging."

For uniformity, we assume that already at the beginning, for all vertices $v$ of $G$, we have the trivial labels $A(v) = \emptyset$, $S(v) = \emptyset$, and $d(v) = 0$ and consider $G$ as a (trivially) *labeled graph*. We frequently deal with pairs of the form $(G, T)$ where $G$ is a labeled graph, and $T$ is a treedepth decomposition of $G$.

### 4.1 Treedepth of Labeled Graphs

The above considerations lead us to the following recursive definition of treedepth for labeled graphs. Let $G = (V, E, d, A, S)$ be a labeled graph. Importantly, when we delete a vertex and obtain $G - v$, the vertex $v$ is also deleted from all the ancestor sets $A(u)$ for all $u \in V(G - v)$.

$$td(G) = \begin{cases} 1 + d(v) & \text{if } |V(G)| = 1; \\ \max_{G' \in C(G)} td(G') & \text{if } |C(G)| > 1; \\ 1 + \min_{v \in V(G) \text{ with } A(v)=\emptyset} \max(td(G - v), d(v)) & \text{otherwise.} \end{cases}$$

Now, $T$ is a treedepth decomposition of the labeled graph $G$ of weighted depth $D$ if all the following properties hold:

T1 $T$ is a treedepth decomposition of the unlabeled graph $(V, E)$,
T2 for every $v \in V$, $d(v) + depth_T(v) \leq D$,
T3 for every $v \in V$ and $u \in A(v)$, $u$ is an ancestor of $v$ in $T$.

Given a labeled graph $G$, the *weighted depth of a rooted tree* $T$, denoted by $depth(T)$, with $V(T) \subseteq V(G)$ is $\max_{v \in V(T)} depth_T(v) + d(v)$. Since our definitions extend smoothly to the unlabeled case by means of trivial labels, we may sometimes refer to "weighted depth" simply as "depth."

### 4.2 Contracting Subtrees

For a labeled graph $G$ and a vertex $v \in V(G)$, we denote the operation of *contracting the subtree* $T_v$ in the decomposition $T$ of graph $G$ by $(G, T) \uparrow v = (G', T')$, yielding a new graph $G'$ and a decomposition $T'$. $G'$ is obtained by identifying the vertices $V(T_v)$ in $G$ and updating the labels as follows: we set $d(v) = depth(T_v)$, we push $T_v$ onto $S(v)$, we add to the set $A(v)$ the set of vertices $u \in V(G) \setminus V(T_v)$ which are adjacent to some $w \in V(T_v) \setminus \{v\}$, and we tag the newly added elements in $A(v)$ with the decomposition $T_v$. $T'$ is obtained by deleting the vertices $V(T_v) \setminus \{v\}$ from the decomposition $T$. It is easy to verify that $T'$ is a treedepth decomposition of $G'$.

### 4.3 Expanding Subtrees

Let $(G', T')$ be a pair consisting of a labeled graph and the corresponding decomposition obtained from $(G, T)$ by a sequence of contractions. Let $v \in V(G')$ be a vertex with a nontrivial label $S(v)$; $v$ is not necessarily a leaf of $T'$. From $(G', T')$ we obtain a labeled graph $G^*$ and a decomposition $T^*$ by the operation of *expanding the vertex $v$ in the decomposition $T'$ of graph $G'$* denoted as $(G', T') \downarrow v = (G^*, T^*)$. $G^*$ is obtained as follows: we delete from $A(v)$ the elements tagged with $T_v$, we pop $T_v$ from $S(v)$, we set $d(v) = depth(last(S(v)))$, and we add to $G'$ all the vertices from $V(T_v) \setminus \{v\}$ and all the edges $uw \in E(G)$ with $u \in V(T_v) \setminus \{v\}$, $w \in V(G) \setminus V(T_v)$. Although the tree $T'$ can be very different from the original $T$ we started with, we can still extend it with $T_v$, just adding it as a subtree of $v$ (possibly next to some existing subtree in $T'$), obtaining a new tree $T^*$.

**Lemma 1.** *If $(G', T') \downarrow v = (G^*, T^*)$ is obtained by the process described above, then $T^*$ is a treedepth decomposition of $G^*$.*

### 4.4 Improving a Subtree

Let $G = (V, E, d, A, S)$ be a labeled graph and $T$ a treedepth decomposition of $G$. Let $v \in V(G)$ and $T_v$ a treedepth decomposition of the induced labeled graph $G[T_v]$. Let $T''$ be a different treedepth decomposition of $G[T_v]$ of weighted depth not exceeding that of $T_v$ (note that the root of $T''$ need not necessarily be $v$). Finally, let $T'$ be the tree obtained from $T$ by replacing $T_v$ by $T''$.

**Lemma 2.** *$T'$ is a treedepth decomposition of $G$ of weighted depth not larger than the weighted depth of $T$.*

### 4.5 The Improvement Procedure

We now provide a high-level overview of the *local improvement procedure* (see Algorithm 1). The procedure takes as input a (possibly labeled) graph $G$, a starting treedepth decomposition $T$ of $G$, and returns a treedepth decomposition $T'$ of $G$ such that $depth(T') \leq depth(T)$. It requires three parameters:

- *budget* $\beta$ which indicates a conservative estimate of the maximum size of instances that is practically feasible for a SAT-solver to solve reasonably quickly (within a few seconds),
- *timeout* $\tau$ (in seconds) for each individual SAT (or MaxSAT) call,
- *contraction size* $\kappa$ which denotes the maximum size of subtrees that can be contracted.

```
Input  : Graph G, decomposition T of G, budget β,
         timeout τ (in seconds), contraction size κ
Output: Decomposition T' of G such that td(T') ≤ td(T)
begin
    (G', T') ⟵ (G, T)
    // Contraction Phase
    repeat
        v ⟵ GetNode(T', β)
        Improve T'_v (either by using SAT-solver or Lemma 4)
        while contraction condition do
            u ⟵ GetNode(T', κ)
            (G', T') ⟵ (G', T') ↑ u              // contract at u
        end
    until r(T) ∈ T'_v
    // Expansion Phase
    while there exists u with nontrivial S(u) do
        (G', T') ⟵ (G', T') ↓ u                  // expand at u
    end
    return T'
end
```

**Algorithm 1.** Pseudocode for TD-SLIM

One individual pass of the procedure consists of a *contraction phase* during which only improvement and contraction operations occur and an *expansion phase* during which the contracted treedepth decomposition is expanded to obtain a treedepth decomposition for the original graph $G$. The contraction phase terminates when we encounter, as the instance to be improved, a local instance containing the root of the starting decomposition $T$. The *contraction condition* on Line 6 determines how many contractions to perform and is only relevant when $\kappa < \beta$. We stop contracting when the size (or depth) of the contracted subinstance falls below a threshold (see partial contraction strategy in Sect. 6.3). The local improvement procedure itself can be repeated any number of times and the potentially improved treedepth decomposition $T'$ returned by one iteration can be used as the starting decomposition for the next iteration. Each individual iteration requires polynomial time in addition to the time required for the SAT (or MaxSAT) call.

At the heart of the contraction phase of the improvement procedure is the `GetNode` subroutine, which is responsible for finding nontrivial sub-instances which can be improved and then contracted. This subroutine takes as input a treedepth decomposition $T$ and a budget $\beta$. It first tries to find a nonleaf vertex $v \in V(T)$ such that $|T_v| \leq \beta$. If no such vertex exists, it instead returns a vertex $u \in V(T)$ such that $height_T(u)$ is 2 and $u$ has at least $\beta$ children, the existence of which is proven by the following lemma.

**Lemma 3.** *Given an integer $k$ and a rooted tree $T$ such that $|T| \geq 2$, at least one of the following conditions is true:*

1. *There exists a nonleaf vertex $v \in V(T)$ such that $|T_v| \leq k$ and $|T_p| > k$ where $p$ is the parent of $v$ in $T$.*
2. *There exists a vertex $u \in V(T)$ such that $height_T(u)$ is 2 and $u$ has at least $k$ children.*

*Proof.* Let us assume for the sake of contradiction that both conditions are false. Let $v$ be a deepest leaf in $T$ and let $u$ be the parent of $v$ in $T$. Note that $height_T(u) = 2$, otherwise $v$ would not be a deepest leaf. Since by our assumption, the second condition is false, $p$ can have at most $k - 1$ children and since its height is 2, $p$'s children are its only descendants. Hence $|T_p| \leq k$, implying that the first condition is true, thus contradicting our assumption. □

When we are unable to find a nonleaf vertex $v$ such that $|T_v| \leq \beta$, it means that there are no subinstances remaining which can be improved and hence the contraction phase of the algorithm would have to terminate abruptly. It is also worth noting that any improvement in the depth of the final decomposition $T'$, as compared the initial decomposition $T$, can be traced back to the contraction phase. Thus, an early termination of this phase means a narrower scope for improvement. But as can be seen in Lemma 3, whenever we are unable to find a reasonably-sized subinstance, we can use this fact—it must be due to a high-degree parent—to our advantage by tackling this case separately.

**Lemma 4.** *Given a labeled graph $G$ which is a star on $n$ vertices, the treedepth of $G$ can be determined in time $\mathcal{O}(n \log n)$.*

## 5 MaxSAT Encoding

Ganian et al. [9], introduced and compared two SAT-encodings for treedepth, one explicitly guessing the tree-structure of a treedepth decomposition and one using a novel partition-based characterization of treedepth, the latter outperforming the former significantly. In both cases, given an unlabeled graph $G$ and an integer $k$, a CNF formula $F(G,k)$ is produced, which is satisfiable if and only if the treedepth of $G$ is at most $k$. By trying out different values of $k$ one can determine the exact treedepth of $G$.

In this section, we build upon Ganian et al.'s [9] partition-based encoding and describe extensions required to employ the SAT encoding for *labeled graphs* thereby addressing the problem of computing treedepth decomposition of weighted graphs with ancestry constraints. We further explain how the encoding can be lifted to MaxSAT, yielding a significant speedup.

### 5.1 Partition-Based Formulation

A *weak partition* of a set $S$ is a set $P$ of nonempty disjoint subsets of $S$. The elements of $P$ are called *equivalence classes*. Let $P, P'$ be two partitions of $S$, then $P'$ is a *refinement* of $P$ if any two elements $x, y \in S$ that are in the same equivalence class of $P'$ are also in the same equivalence class of $P$. Given a set $S$, a *derivation* $\mathcal{P}$ of *length* $\ell$ is a sequence $(P_1, P_2, \ldots, P_\ell)$ of weak partitions of $S$. $P_i$ is called the *$i$-th level* of $\mathcal{P}$. For some $2 \leq i \leq \ell$, we say that a set $c \in P_{i-1}$ is a *child* of a set $p \in P_i$ if $c \subseteq p$. We denote by $\mathsf{c}^i_{\mathcal{P}}(p)$ the set of all children of $p$ at level $i$. Further, $\chi^i_{\mathcal{P}}(p)$ denotes the set $p \setminus \bigcup_{c \in \mathsf{c}^i_{\mathcal{P}}(p)} c$. Finally, the shorthand $\bigcup P_i$ denotes the set $\bigcup_{p \in P_i} p$. Given a labeled graph $G$, a *derivation $\mathcal{P}$ of $G$* is a sequence $(P_1, \ldots, P_\ell)$ of weak partitions of the set $V(G)$ satisfying the following properties:

D1 $P_1 = \emptyset$ and $P_\ell = \{V(G)\}$;
D2 for every $1 \leq i \leq \ell - 1$, $P_i$ is a refinement of $P_{i+1}$;
D3 for every $1 \leq i \leq \ell$ and $p \in P_i$, $|\chi^i_{\mathcal{P}}(p)| \leq 1$;
D4 for every edge $uv \in E(G)$, there is a $p \in P_i$ for some $1 \leq i \leq \ell$ such that $u, v \in p$ and $\chi^i_{\mathcal{P}}(p) \cap \{u, v\} \neq \emptyset$;
D5 for every $v \in V(G)$ and $1 \leq i \leq \ell$, if $v \in \bigcup P_i$ then $d(v) + 2 \leq i$; and
D6 for every $v \in V(G)$ and $u \in A(v)$, there is a $p \in P_i$ for some $1 \leq i \leq \ell$ such that $u, v \in p$ and $u \in \chi^i_{\mathcal{P}}(p)$; together with D3 that implies $\chi^i_{\mathcal{P}}(p) = \{u\}$.

**Theorem 1.** *Let $G = (V, E, d, A, S)$ be a labeled graph and $D$ an integer. $G$ has a treedepth decomposition of weighted depth at most $D$ if and only if $G$ has a derivation of length at most $D + 1$.*

*Proof.* Let $T$ be a treedepth decomposition of $G$ of weighted depth $D$. Let $\mathcal{P}$ be the derivation consisting of weak partitions $(P_1, \ldots, P_{D+1})$ where $P_1 = \emptyset$, $P_i = \{\, V(T_u) \mid u \in V(T) \text{ and } \mathit{depth}_T(u) = D - i + 2 \,\}$ for every $2 \leq i \leq D + 1$. It is easy to see that $\mathcal{P}$ is a derivation of the unlabeled graph $(V, E)$ and the

length of $\mathcal{P}$ is $D+1$. For any vertex $v \in V(G)$, let $2 \leq i \leq D+1$ such that $v \in \bigcup P_i$, thus, by construction of $\mathcal{P}$, we get $depth_T(v) \geq D - i + 2$. Since $T$ is a treedepth decomposition of the labeled graph $G$, it satisfies property T2, meaning $D \geq d(v) + depth_T(v) \geq d(v) + D - i + 2$ which implies $i \geq d(v) + 2$, which is precisely property D5. To show property D6, let $v \in V(G)$ and $u \in A(v)$, let $k = D - depth_T(u) + 2$. We observe that there exists $p \in P_k$ such that $u \in p$, and since $u$ is an ancestor of $v$ (from property T3), $v \in T_u$ therefore $v \in p$. We further observe that $u \notin \bigcup P_{k-1}$ meaning $u \in \chi^k_{\mathcal{P}}(p)$, hence property D6 holds.

Towards showing the converse, let $\mathcal{P}$ be a derivation of length $D+1$ of the labeled graph $G$. Note that w.l.o.g we can assume $\chi^i_{\mathcal{P}}(p) \neq \emptyset$ for every $1 \leq i \leq D+1$ and $p \in P_i$, because otherwise we could replace $p$ with all its children without increasing the length of the derivation and retaining all the properties. For every $v \in V(G)$ there exists exactly one $1 \leq i \leq D+1$ and $p \in P_i$ such that $\chi^i_{\mathcal{P}}(p) = \{v\}$; in that case we say that the set $p$ *introduces* $v$. Now we construct the treedepth decomposition $T$ with vertex set $V(G)$ by adding an edge between $u, v \in V(G)$ if the set introducing $u$ is a child of the set introducing $v$ or vice versa. It can be seen that $T$ is a treedepth decomposition of depth at most $D$ of the unlabeled graph $(V, E)$.

Towards showing T2, let $v \in V(G)$ and $1 < i \leq D+1$ such that $v \in \bigcup P_i$ and $v \notin \bigcup P_{i-1}$; in other words, $i$ is the smallest index such that $v \in \bigcup P_i$. By construction of $T$, this implies that $v$ is introduced in the $i$-th layer, meaning $depth_T(v) = D - i + 2$. Combining this with property D5, we get $d(v) + depth_T(v) \leq D$, thus satisfying property T2. Now, to show T3, since $\mathcal{P}$ satisfies property D6, let $2 \leq i \leq D+1$ such that $u, v \in p$ for some $p \in P_i$ and $\chi^i_{\mathcal{P}}(p) = \{u\}$. Thus $p$ introduces $u$. Further, $v \in \mathsf{c}^i_{\mathcal{P}}(p)$ since $v \notin \chi^i_{\mathcal{P}}(p)$, meaning $u$ is an ancestor of $v$ in $T$, therefore satisfying property T3. Hence $T$ is indeed a treedepth decomposition of the labeled graph $G$. □

We note that the treedepth of a labeled graph $G = (V, E, d, A, S)$ can be as large as $|V| + \max_{v \in V} d(v)$. Since the above proof explicitly encodes the weight labels $d(v)$ in the derivation, the treedepth $D$ affects the number of layers in the derivation, which in turn affects the size of the encoding. Thus, for graphs with large $d(v)$, the encoding size is also large. A neat observation can however remedy this: the derivation of $G$ does not need to have any more than $|V| + 1$ layers.

**Observation 1.** *Let $G = (V, E, d, A, S)$ be a labeled graph and $D > |V|$ be an integer. Let $G' = (V, E, d', A, S)$ where $d'(v) := \max(0, d(v) - (D - |V|))$ for $v \in V(G)$. $G$ has a derivation of length at most $D+1$ if and only if $G'$ has a derivation of length at most $|V| + 1$.*

### 5.2 Encoding of a Derivation

We now tersely describe the encoding of the formulation discussed in the previous subsection.

**Theorem 2.** *Given a labeled graph $G$ and an integer $D$, one can construct in polynomial time, a CNF formula $F(G, D)$ that is satisfiable if and only if $G$ has a derivation of length at most $D$.*

The remainder of this section describes the clauses constituting the formula $F$. We have a set variable $s(u, v, i)$, for every $u, v \in V(G)$ with $u \leq v$ and every $i$ with $1 \leq i \leq D$. The variable $s(u, v, i)$ indicates whether vertices $u$ and $v$ appear in the same equivalence class of $P_i$, and $s(u, u, i)$ indicates whether $u$ appears in some equivalence class of $P_i$. The following clauses ensure D1 and D2:

$$\neg s(u, v, 1) \wedge s(u, v, D) \qquad \text{for } u, v \in V(G), u \leq v, \text{ and}$$
$$\neg s(u, v, i) \vee s(u, v, i + 1) \qquad \text{for } u, v \in V(G), u \leq v, 1 \leq i \leq D.$$

The following clauses ensure D3 and D4:

$$\neg s(u, v, i) \vee s(u, u, i - 1) \vee s(v, v, i - 1) \qquad \text{for } u, v \in V(G), u < v, 2 \leq i \leq D,$$
$$\neg s(u, u, i) \vee \neg s(v, v, i) \vee s(u, u, i - 1) \vee s(u, v, i)$$
$$\neg s(u, u, i) \vee \neg s(v, v, i) \vee s(v, v, i - 1) \vee s(u, v, i)$$
$$\text{for } uv \in E, u < v, 2 \leq i \leq D.$$

The following clauses ensure the semantics of the set variables of the form $s(u, u, i)$ as well as the transitivity of the set variables:

$$(\neg s(u, v, i) \vee s(u, u, i)) \wedge (\neg s(u, v, i) \vee s(v, v, i))$$
$$\text{for } u, v \in V(G), u < v, 2 \leq i \leq D,$$
$$(\neg s(u, v, i) \vee \neg s(u, w, i) \vee s(v, w, i))$$
$$\wedge(\neg s(u, v, i) \vee \neg s(v, w, i) \vee s(u, w, i))$$
$$\wedge(\neg s(u, w, i) \vee \neg s(v, w, i) \vee s(u, v, i))$$
$$\text{for } u, v, w \in V(G), u < v < w, 1 \leq i \leq D.$$

Finally, the following clauses ensure D5 and D6:

$$\neg s(u, u, i) \quad \text{for } u \in V(G), 2 \leq i \leq D, \text{if } |V(G)| + d(u) \geq D \text{ and } i < d(u) + 2,$$
$$\neg s(v, v, i) \vee s(u, u, i) \qquad \text{for } u \in V(G), v \in A(u) \text{ and } 2 \leq i \leq D.$$

This concludes the description of the SAT encoding. Note that, due to Observation 1, without loss of generality, we may assume that $D \leq |V| + 1$.

We now extend the above encoding to a Partial MaxSAT formulation $F'(G, D)$ containing soft clauses. An optimal solution for $F'(G, D)$ satisfies $\mu$ soft clauses if and only if $G$ has treedepth $D - \mu + 1$. First, all the clauses from the SAT encoding are added as hard clauses into $F'$. Then we introduce a *free layer* variable $f_i$ for $1 \leq i \leq D$, which is false if some vertex appears in the $i$-th layer, i.e., $u \in \bigcup P_i$ for some $u \in V(G)$. Further, if the $i$-th layer is free then all the lower layers must also be free. These conditions are encoded via the following hard clauses:

$$\neg f_i \vee \neg s(u,u,i) \qquad \text{for } u \in V(G) \text{ and } 2 \leq i \leq D,$$
$$\neg f_i \vee f_{i-1} \qquad \text{for } 2 \leq i \leq D.$$

Additionally, we need to take special care in the case of labeled graphs, as the depth labels could mean that even though no vertices appear in a layer, they are still occupied by a subtree represented by the depth labels. In other words, a vertex $v$ must not only "occupy" its own layer but also $d(v)$ many layers below its own layer. The following hard clause captures this condition:

$$\neg s(u,u,i) \vee \neg f_{i-j} \qquad \text{for } u \in V(G), 2 \leq i \leq D \text{ and } 1 \leq j \leq \min(d(u), i).$$

Finally, we introduce a soft unit clause $f_i$ for $1 \leq i \leq D$. This sets the objective of the MaxSAT solver to maximize the number of free layers, which consequently, minimizes the depth of the decomposition. This concludes the description of the MaxSAT encoding.

## 6 Experimental Evaluation

### 6.1 Experimental Setup

We ran all our experiments on a 10-core Intel Xeon E5-2640 v4, 2.40 GHz CPU, with each process having access to 8 GB RAM. We used Glucose 4.0[2] as the SAT-solver and UWrMaxSat as the MaxSAT-solver, both with standard settings. We use UWrMaxSat primarily due to its anytime nature (i.e., it can be terminated anytime, and it outputs the current best possibly suboptimal result). We also tried other solvers like RC2 and Loandra from the 2019 MaxSAT Evaluation contest, but UWrMaxSat worked better for our use case. The details of all the MaxSAT solvers can be found on the 2019 MaxSAT Evaluation webpage[3].

We implemented the local improvement algorithm in Python 3.6.9, using the Networkx 2.4 graph library [12] and the PySAT 0.1.5 library [14] for the MaxSAT solver RC2. The source code of our implementation is available online [32]. Our experiments aim to demonstrate the benefit of applying local improvement to any external heuristic, and not to provide a comparison between our approach coupled with the two considered heuristics and other standalone algorithms.

### 6.2 Instances

We tested our implementation on subsets of the public benchmark instances used by the PACE Challenge 2020, from both the Exact Track (smaller instances on average) and the Heuristic Track. We formed two datasets as follows: (i) *Dataset A* consists of all the instances on which the heuristic algorithms were able to compute a solution within 2 h. This yielded 140 instances in the range $|V| \in [10, 4941]$, $|E| \in [15, 86528]$. This dataset is meant to serve as a

[2] https://www.labri.fr/perso/lsimon/glucose/.
[3] https://maxsat-evaluations.github.io/2019/descriptions.html.

comprehensive dataset with a large variance in the graph sizes. (ii) *Dataset B* consists of 30 instances from the Exact Track (27–85). This dataset represents the set of instances that we expect to lie in the practically feasible zone of SAT-solvers (or MaxSAT solvers). The resulting graphs lie in the range $|V| \in [30, 72]$, $|E| \in [48, 525]$.

### 6.3 Experiment 1

We evaluate the quality of the solution in terms of the improvement in depth, where *absolute improvement* (or simply *improvement*) refers to the difference between the starting heuristic depth and the final reported depth, and *relative improvement* (RI) refers to the absolute improvement expressed as a percentage of the starting heuristic depth. Our implementation of the algorithm can be configured using the following parameters:

- budget $\beta$, can either be a single value or a sequence of budget values (we denote by *Multibudget* the sequence $(5+(5i \bmod 40))_{i\geq 0}$ where the next budget value is used when the current value fails to provide any improvement),
- timeout $\tau$,
- contraction ratio $\gamma$ which determines the contraction size $\kappa = \gamma\beta$,
- partial contraction strategy which determines when we switch from contraction to improvement—either when the local instance's size has been reduced by half or when the local instance's unweighted depth has been reduced by 2,
- target solver, i.e., SAT or MaxSAT,
- random seed, and
- global timeout.

Since the number of possible parameter configurations is huge, we first ran a preliminary experiment on a small number of instances with a large number of parameter configurations to narrow down the better performing configurations. We gathered a smaller set of configurations from this initial run, which we then tested rigorously on Dataset A. We tried $\beta \in \{20, \text{Multibudget}\}$, $\tau = 20$, $\gamma \in \{0.5, 1\}$, partial contraction by depth, target solver MaxSAT, and random seed $\in \{1, 2, 3\}$. For the starting decomposition, we compared two heuristics proposed by Villaamil [42]—a randomized variant of the DFS heuristic and one of the separator-based heuristics (denoted by Sep). Given a treedepth heuristic algorithm X, we denote by TD-SLIM(X) the algorithm obtained by running the local improvement procedure on top of the heuristic solution provided by algorithm X. The implementation of Sep was kindly provided to us by Oelschlägel [31]. We precomputed the heuristic solutions for Dataset A with a 2-h timeout and then ran the improvement procedure for 30 min.

We use three solvers or configurations to present the performance data and convey the likelihood of the different results:

- *Virtual Best Solver* (VBS): the hypothetical solver which, for each instance, knows the configuration that yields the best improvement,

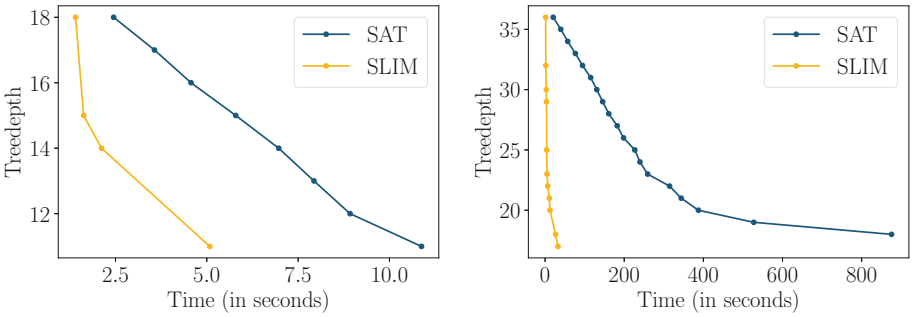
- *Single Best Solver* (SBS): the solver with the configuration that resulted in the best improvement on average across all instances,
- *Average Solver* (AS): the hypothetical solver representing the average performance across all the configurations for a particular instance.

The average relative improvements (including the cases with no improvement) for AS, SBS, and VBS were 45.9%, 52.2%, and 53.0% when starting from the DFS heuristic and 21.9%, 29.2%, and 30.2% when starting from the Sep heuristic, respectively. Table 1 shows the instances from Dataset A with the best relative improvement. The parameter combination to achieve the best average relative improvement across all the 140 instances across both heuristics (i.e., SBS) was (Multibudget, MaxSAT, $\gamma = 0.5$, partial contraction by depth). In the experiment, we observed a rather robust performance over all considered configurations.

**Table 1.** Top 15 instances from Experiment 1 sorted by best relative improvement among both heuristics. *Start* and *Final* refer to the starting heuristic depth and the final reported depth, respectively *Imp.* refers to the absolute improvement.

| Instance | $\|V\|$ | $\|E\|$ | DFS heuristic | | | | Sep heuristic | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Start | Final | Imp. | RI (%) | Start | Final | Imp. | RI (%) |
| heur_055 | 590 | 668 | 92 | 18 | 74 | 80.43 | 27 | **17** | 10 | 37.04 |
| heur_033 | 255 | 507 | 129 | **34** | 95 | 73.64 | 155 | **34** | 121 | 78.06 |
| heur_071 | 1023 | 2043 | 486 | **108** | 378 | 77.78 | 657 | 418 | 239 | 36.38 |
| exact_165 | 176 | 186 | 49 | 11 | 38 | 77.55 | 17 | **10** | 7 | 41.18 |
| exact_193 | 449 | 2213 | 128 | **29** | 99 | 77.34 | 151 | 75 | 76 | 50.33 |
| heur_021 | 195 | 340 | 100 | 23 | 77 | 77.00 | 43 | **22** | 21 | 48.84 |
| exact_173 | 198 | 692 | 69 | **16** | 53 | 76.81 | 57 | 50 | 7 | 12.28 |
| exact_169 | 181 | 253 | 77 | **18** | 59 | 76.62 | 44 | 20 | 24 | 54.55 |
| exact_195 | 451 | 587 | 131 | 31 | 100 | 76.34 | 74 | **23** | 51 | 68.92 |
| exact_157 | 163 | 195 | 49 | **12** | 37 | 75.51 | 27 | 13 | 14 | 51.85 |
| exact_103 | 92 | 131 | 57 | **14** | 43 | 75.44 | 24 | 16 | 8 | 33.33 |
| heur_025 | 212 | 257 | 67 | **17** | 50 | 74.63 | 36 | 19 | 17 | 47.22 |
| exact_177 | 204 | 248 | 34 | **9** | 25 | 73.53 | 16 | 11 | 5 | 31.25 |
| exact_107 | 95 | 121 | 41 | **11** | 30 | 73.17 | 23 | 13 | 10 | 43.48 |
| exact_185 | 276 | 1187 | 63 | **20** | 43 | 68.25 | 85 | 23 | 62 | 72.94 |

Out of the 140 instances solved by both heuristics, TD-SLIM(DFS) provided a strictly lower depth than TD-SLIM(Sep) for 48 instances and strictly higher depth for 38 instances. In the remaining 54 instances, both TD-SLIM(DFS) and TD-SLIM(Sep) reached the same depth value. Thus, TD-SLIM is often capable of achieving a comparable or even lower depth despite starting from a significantly




(a) exact_031.gr (b) dolphin.gml

**Fig. 2.** TD-SLIM vs SAT

worse heuristic decomposition, but there are also cases that show that TD-SLIM is capable of utilizing and building upon a better starting decomposition. Comparing these depths with the lowest known depths from the PACE challenge, TD-SLIM(DFS) matches the lowest depth on 53 instances and is off by one on 20 instances. TD-SLIM(Sep) matches the lowest depth on 41 instances and is off by one on 17 instances.

Very recently, the results for the PACE challenge were announced and the implementations of the solvers were made available. Consequently, we tested TD-SLIM on top of the winning heuristic ExTREEm [41]. We ran the heuristic for 15 min and then TD-SLIM on top of the solution provided by the heuristic for 15 min, using the same total time limit of 30 min as used in the PACE challenge. Somewhat surprisingly, we observed that for 6 instances, TD-SLIM(ExTREEm) was able to compute better decompositions than simply running ExTREEm for 30 min. For 3 of these instances, TD-SLIM(ExTREEm) was even able to find a better decomposition than all the 55 participating heuristic solvers.

### 6.4 Experiment 2

We tested the effectiveness of TD-SLIM over a one-shot SAT or MaxSAT encoding where the entire instance is passed to the solver. To give the one-shot SAT or MaxSAT a massive advantage, we chose Dataset B, which contains much smaller instances, and we set the global timeout for TD-SLIM to 30s, whereas for SAT and MaxSAT we chose 300s. For this experiment, we used DFS as the starting heuristic. We observed that TD-SLIM performs comparably to SAT in terms of the final depth achieved, computing the best upper bound for 18 instances while SAT arrives at the best upper bound for 22 instances. Nevertheless, when it comes to slightly larger instances, TD-SLIM even outperforms SAT in 7 instances despite only having a tenth of the time. We also noticed that MaxSAT tends to perform significantly worse than SAT. We suspect this

might be the case because the MaxSAT solver spends time improving the lower bounds, which is of little use in this case.

As a part of Experiment 2, we also compared the trajectory of improvement over time of TD-SLIM and SAT. We use graphs 'exact_031.gr' and 'dolphin.gml' (from [27]) as typical examples. As can be seen in Fig. 2a and Fig. 2b, TD-SLIM is much faster than SAT. Another interesting observation is that TD-SLIM is able to achieve depth values which were previously not possible, e.g., for the graph 'B10Cage', TD-SLIM improved the previously known upper bound from 23 [9] to 22 with a runtime of around 50s.

## 7 Concluding Remarks

Our (Max)SAT-based local improvement approach to treedepth provides a compelling showcase for demonstrating how (Max)SAT encodings can be scaled to large inputs, thus widening the scope of potential applications for exact constraint-based methods. Our experiments show that in many cases, our approach allows a significant improvement over heuristically obtained treedepth decompositions. We observed that TD-SLIM is able to improve even over strong heuristics like the winning heuristic from the PACE challenge. Rather unexpected is the finding that on smaller instances, the local improvement method significantly outperforms a one-shot (Max)SAT encoding.

In future work, we plan to systematically study the effect of postprocessing for different types of heuristics, as have been made available by the PACE challenge. Other topics for future work include the utilization of incremental solving techniques for the SAT-based optimization of treedepth, and its comparison to the MaxSAT approach, as well as the development of symmetry breaking methods for further speeding up the encoding.

## References

1. Bannach, M., Berndt, S., Ehlers, T.: Jdrasil: a modular library for computing tree decompositions. In: Iliopoulos, C.S., Pissis, S.P., Puglisi, S.J., Raman, R. (eds.) 16th International Symposium on Experimental Algorithms, SEA 2017, London, UK, 21–23 June 2017, vol. 75, pp. 28:1–28:21. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
2. Berg, J., Järvisalo, M.: SAT-based approaches to treewidth computation: an evaluation. In: 26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, 10–12 November 2014, pp. 328–335. IEEE Computer Society (2014)
3. Dechter, R., Mateescu, R.: AND/OR search spaces for graphical models. Artif. Intell. **171**(2–3), 73–106 (2007)
4. Fichte, J.K., Hecher, M., Lodha, N., Szeider, S.: An SMT approach to fractional hypertree width. In: Hooker, J. (ed.) CP 2018. LNCS, vol. 11008, pp. 109–127. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98334-9_8

5. Fichte, J.K., Lodha, N., Szeider, S.: SAT-based local improvement for finding tree decompositions of small width. In: Gaspers, S., Walsh, T. (eds.) SAT 2017. LNCS, vol. 10491, pp. 401–411. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66263-3_25
6. Fomin, F.V., Giannopoulou, A.C., Pilipczuk, M.: Computing tree-depth faster than $2^n$. Algorithmica **73**(1), 202–216 (2015)
7. Freuder, E.C., Quinn, M.J.: Taking advantage of stable sets of variables in constraint satisfaction problems. In: IJCAI, vol. 85, pp. 1076–1078. Citeseer (1985)
8. Gajarský, J., Hlinený, P.: Faster deciding MSO properties of trees of fixed height, and some consequences. In: D'Souza, D., Kavitha, T., Radhakrishnan, J. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, Hyderabad, India, 15–17 December 2012, vol. 18, pp. 112–123. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)
9. Ganian, R., Lodha, N., Ordyniak, S., Szeider, S.: SAT-encodings for treecut width and treedepth. In: Kobourov, S.G., Meyerhenke, H. (eds.) Proceedings of ALENEX 2019, the 21st Workshop on Algorithm Engineering and Experiments, pp. 117–129. SIAM (2019)
10. Gutin, G., Jones, M., Wahlström, M.: Structural parameterizations of the mixed Chinese postman problem. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 668–679. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_56
11. Hafsteinsson, H.: Parallel sparse Cholesky factorization. Cornell University, Technical report (1988)
12. Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using NetworkX. In: Proceedings of the 7th Python in Science Conference (SciPy 2008), Pasadena, CA, USA, pp. 11–15, August 2008
13. Heule, M., Szeider, S.: A SAT approach to clique-width. ACM Trans. Comput. Log. **16**(3), 24 (2015). https://doi.org/10.1145/2736696
14. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: a Python toolkit for prototyping with SAT Oracles. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 428–437. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_26
15. Iwata, Y., Ogasawara, T., Ohsaka, N.: On the power of tree-depth for fully polynomial FPT algorithms. In: Niedermeier, R., Vallée, B. (eds.) 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, 28 February–3 March 2018, Caen, France, vol. 96, pp. 41:1–41:14. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018)
16. Iyer, A.V., Ratliff, H.D., Vijayan, G.: On a node ranking problem of trees and graphs. Technical reporrt, Georgia Inst of Tech Atlanta Production and Distribution Research Center (1986)
17. Jégou, P., Terrioux, C.: Hybrid backtracking bounded by tree-decomposition of constraint networks. Artif. Intell. **146**(1), 43–75 (2003)
18. Jess, J.A.G., Kees, H.G.M.: A data structure for parallel L/U decomposition. IEEE Trans. Comput. **3**, 231–239 (1982)
19. Kayaaslan, E., Uçar, B.: Reducing elimination tree height for parallel LU factorization of sparse unsymmetric matrices. In: 2014 21st International Conference on High Performance Computing (HiPC), pp. 1–10. IEEE (2014)
20. Kees, H.G.M.: The organization of circuit analysis on array architectures. Ph.D. thesis, Citeseer (1982)
21. Liu, J.W.: Reordering sparse matrices for parallel elimination. Parallel Comput. **11**(1), 73–91 (1989)

22. Liu, J.W.: The role of elimination trees in sparse factorization. SIAM J. Mat. Anal. Appl. **11**(1), 134–172 (1990)
23. Llewellyn, D.C., Tovey, C., Trick, M.: Local optimization on graphs. Discrete Appl. Math. **23**(2), 157–178 (1989)
24. Lodha, N., Ordyniak, S., Szeider, S.: A SAT approach to branchwidth. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 179–195. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_12
25. Lodha, N., Ordyniak, S., Szeider, S.: SAT-encodings for special treewidth and pathwidth. In: Gaspers, S., Walsh, T. (eds.) SAT 2017. LNCS, vol. 10491, pp. 429–445. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66263-3_27
26. Lodha, N., Ordyniak, S., Szeider, S.: A SAT approach to branchwidth. ACM Trans. Comput. Log. **20**(3), 15:1–15:24 (2019)
27. Lusseau, D., Schneider, K., Boisseau, O.J., Haase, P., Slooten, E., Dawson, S.M.: The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. Behav. Ecol. Sociobiol. **54**(4), 396–405 (2003)
28. Manne, F.: Reducing the height of an elimination tree through local recorderings. University of Bergen, Department of Informatics (1991)
29. Nesetril, J., de Mendez, P.O.: Tree-depth, subgraph coloring and homomorphism bounds. Eur. J. Comb. **27**(6), 1022–1041 (2006)
30. Nešetřil, J., de Mendez, P.O.: Sparsity - Graphs, Structures, and Algorithms. Algorithms and Combinatorics, vol. 28. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27875-4
31. Oelschlägel, T.: Treewidth from Treedepth. Ph.D. thesis, RWTH Aachen University (2014)
32. Peruvemba Ramaswamy, V., Szeider, S.: aditya95sriram/td-slim: public release, July 2020. Zenodo. https://doi.org/10.5281/zenodo.3946663
33. Peruvemba Ramaswamy, V., Szeider, S.: Turbocharging treewidth-bounded Bayesian network structure learning (2020). https://arxiv.org/abs/2006.13843
34. Pieck, J.: Formele definitie van een e-tree. Eindhoven University of Technology: Department of Mathematics: Memorandum 8006 (1980)
35. Pisinger, D., Ropke, S.: Large neighborhood search. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, pp. 399–419. Springer, Boston (2010). https://doi.org/10.1007/978-1-4419-1665-5_13
36. Pothen, A.: The complexity of optimal elimination trees. Technical report (1988)
37. Pothen, A., Simon, H.D., Wang, L., Barnard, S.T.: Towards a fast implementation of spectral nested dissection. In: Supercomputing 1992: Proceedings of the 1992 ACM/IEEE Conference on Supercomputing, pp. 42–51. IEEE (1992)
38. Reidl, F., Rossmanith, P., Villaamil, F.S., Sikdar, S.: A faster parameterized algorithm for treedepth. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 931–942. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_77
39. Samer, M., Veith, H.: Encoding treewidth into SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 45–50. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_6
40. Schidler, A., Szeider, S.: Computing optimal hypertree decompositions. In: Blelloch, G., Finocchi, I. (eds.) Proceedings of ALENEX 2020, the 22nd Workshop on Algorithm Engineering and Experiments, pp. 1–11. SIAM (2020)
41. Swat, S.: swacisko/pace-2020: first release of ExTREEm, June 2020. Zenodo. https://doi.org/10.5281/zenodo.3873126
42. Villaamil, F.S.: About treedepth and related notions. Ph.D. thesis, RWTH Aachen University (2017)