On Tamaki's Algorithm to Compute Treewidths

Ernst Althaus
Johannes Gutenberg-Universität Mainz, Germany

Daniela Schnurbusch
Johannes Gutenberg-Universität Mainz, Germany

Julian Wüschner
Johannes Gutenberg-Universität Mainz, Germany

Sarah Ziegler
Johannes Gutenberg-Universität Mainz, Germany

Abstract

We revisit the exact algorithm to compute the treewidth of a graph of Tamaki and present it in a way that facilitates improvements. The so-called I-blocks and O-blocks enumerated by the algorithm are interpreted as subtrees of a tree-decomposition that is constructed. This simplifies the proof of correctness and allows to discard subtrees from the enumeration by some simple observations. In our experiments, we show that one of these modifications in particular reduces the number of enumerated objects considerably.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Tree Decomposition, Exact Algorithm, Algorithms Engineering

Digital Object Identifier 10.4230/LIPIcs.SEA.2021.9

Supplementary Material *Software*: `https://gitlab.rlp.net/daschnur/compute_treewidths`
archived at `swh:1:dir:083d68c2152e2521bcd065be491c7a6b35267cc5`

1 Introduction

Tree decompositions are a major tool for obtaining parameterized algorithms for hard graph problems as many problems allow parameterization with respect to the treewidth of the graph (see, e.g., [11]). In order to use such algorithms, the first step is to compute a tree decomposition with minimal width or an approximation thereof.

It is well known that computing an optimal tree decomposition is $\mathcal{NP}$-hard [3]. Owing to the early results of Robertson and Seymour, we know that an optimal tree decomposition can be computed in $\mathcal{O}(n^2)$ if the treewidth is bounded, but the proof is non-constructive. In [5], Bodlaender presented a linear-time algorithm for this problem, effectively settling the matter from a theoretical point of view. Since the description of the algorithm is hardly accessible, we recently published a simpler description of it [1, 2].

The main problem of Bodlaender's algorithm is its running-time dependency on the treewidth. More precisely, for a graph with treewidth tw, the best running time estimate is $2^{\mathcal{O}(tw^3)} \cdot n$, which makes the dependence on the treewidth in the computation of the tree decomposition a major theoretical bottleneck of the computation. Notice that algorithms using tree decompositions do not necessarily require the latter to be optimal for being a parameterized algorithm, as long as their width is bounded in the optimal treewidth. For this reason, efforts to compute *approximately* optimal tree decompositions gained considerable attention in the recent past, leading, among other things, to the discovery of the first such algorithm with linear time dependency on the size of the graph and a single exponential dependency on the treewidth [6].

From an applied perspective, many heuristics were designed to compute tree decompositions, including the min-fill and min-degree heuristics, see, e.g., [15, 12]. Nevertheless, there remains a large interest in practically feasible algorithms for the construction of

optimal tree decompositions, as evidenced, e.g., by recent iterations of the PACE competition (`https://pacechallenge.org`). Currently, the algorithm of Tamaki [15, 14] (`https://github.com/TCS-Meiji/PACE2017-TrackA`) seems to be one of the fastest algorithms.

One key ingredient for practically efficient algorithms is a suitable preprocessing of the (input) graph. Studies have shown that the notion of so-called safe separators is very beneficial for tree decompositions[7]. Safe separators are subsets of the nodes whose removal separates the graph into several parts. An optimal tree decomposition can then be found by considering the components (for each component, including the separator) individually. Bodlaender et al. [7] showed some sufficient conditions for separators to be safe. Furthermore, the authors showed that separators that are an almost clique are safe and presented an algorithm find them. Tamaki [14] gives a heuristic to compute safe separators. The author first computes separators by using a simple heuristic algorithm to compute a tree decomposition and uses the intersections of neighboring bags as candidate separators. For each candidate, he uses an additional heuristic to test another sufficient condition for safe separators.

Computing lower bounds is a subroutine in many exact algorithms for $\mathcal{NP}$-hard problems. An overview of lower bounds for the treewidth is given in [9]. A simple lower bound is obtained by heuristically computing a minor of the given graph and using the second smallest degree as lower bound (if the smallest degree appears at least twice, this degree is to be taken).

In this paper we describe the algorithm of Tamaki in a way that can be interpreted more intuitively. In order to construct an optimal tree decomposition, we assume the tree decomposition to be rooted at a defined vertex. Then we build the tree decomposition from the leaves and call the constructed structures *partial tree decompositions* (a formal definition is given in Section 3). The key insight of the algorithm of Tamaki is that there are only a linear number of possible bags of the root of a partial tree decomposition, knowing the partial tree decompositions for the children. The same holds for the leaves.

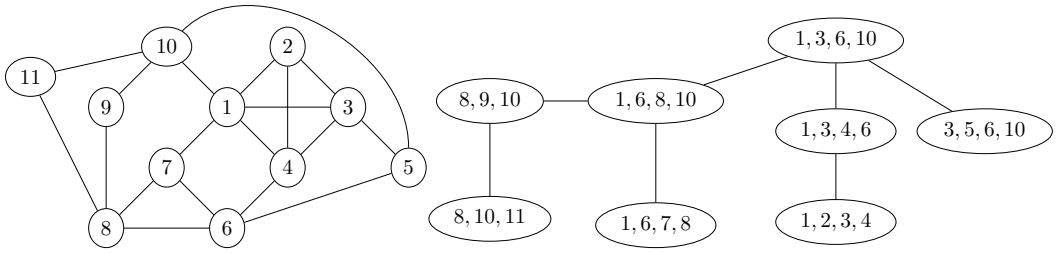
The interpretation of the enumerated partial tree decompositions allows to remove some of them by simple observations that exclude partial tree decompositions from being extended to an optimal tree decomposition or being replaced by partial tree decompositions already enumerated. Some of these observations are automatically guaranteed in the original description of the Tamaki's algorithm. Furthermore, we improve the usage of safe separators by testing the sufficient conditions on all separators constructed when enumerating partial tree decompositions.

The paper is organized as follows: In Section 2, we introduce the terminology, basic definitions, and known properties. The basic algorithm is stated in Section 3, followed by a section on the methods to eliminate partial tree decompositions. Before we show some experimental results in Section 6, we present some additional details on the algorithm in Section 5. Finally, we give a short conclusion.

2 Definitions and Basic Properties

Unless stated otherwise, we assume all graphs to be undirected, finite, and without self-loops. Let G be a graph. We denote its vertex and edge sets as $V(G)$ and $E(G)$, respectively. Additionally, we define

- $N_G(v) := \{u \in V(G) | (u, v) \in E(G)\}$, the open neighborhood of a vertex v in G,
- $N_G[v] := N_G(v) \cup \{v\}$, the closed neighborhood of v in G,
- $deg(v) := |N_G(v)|$, the degree of v in G,
- $N_G(U) := \bigcup_{u \in U} N_G(u) \setminus U$, the open neighborhood of the vertex set $U \subseteq V(G)$,
- $N_G[U] := N_G(U) \cup U$, the closed neighborhood of $U \subseteq V(G)$.



(a) An example graph. **(b)** Corresponding tree decomposition.

Figure 1 Example for a tree decomposition of width 3 from a given graph. In this case, the tree decomposition is optimal and thus also has a treewidth of 3.

The subscript G can be omitted in cases where there is no ambiguity about the graph.

For $U \subseteq V(G)$, $G[U]$ denotes the subgraph of G induced by U, i.e., the graph with vertex set $V(G[U]) = U$ and edge set $E(G[U]) = \{(a, b) \in E(G) | a, b \in U\}$. $G\langle U \rangle$ denotes the graph obtained from G by completing U to a clique.

Let $C, S \subseteq V(G)$ be two vertex sets. We define:

- C is *connected* in G if there exists a path between all pairs of vertices $u, v \in C$,
- C is a *connected component* of G if C is connected and inclusion-wise maximal with this property.
- C is a *component associated with* S if C is a connected component in $G[V(G) \setminus S]$
- S is a *separator* of G if there exist at least two components associated with S.
- S is a *u, v-separator* for vertices $u, v \in V(G)$ if u and v lie in different components associated with S.
- C is a *full component* associated with S if $N(C) = S$.
- S is a *minimal separator* if there exist at least two full components associated with S.

One can show that a minimal separator is minimal in the sense that, if we remove a vertex from it, it no longer separates vertices that lie in different full components. It may, however, still separate vertices in a previously non-full component from the rest of the graph.

▶ **Definition 1** (Tree Decomposition, [13]). *For a graph G, let T be a tree and $(X_t)_{t \in V(T)}$ a family of vertex sets indexed by the nodes of T with $X_t \subseteq V(G)$ for all $t \in V(G)$. $\mathcal{T} := (T, (X_t)_{t \in V(T)})$ is called a tree decomposition of G if*

1. $\bigcup_{t \in V(T)} X_t = V$,
2. *for each edge $e = (u, v) \in E(G)$, there exists a node $t \in V(T)$ with $u, v \in X_t$, and*
3. *for all $t_1, t_2, t_3 \in V(T)$, such that node t_2 lies on the path between nodes t_1 and t_3, $X_{t_1} \cap X_{t_3} \subseteq X_{t_2}$.*

We will refer to the third condition as the *consistency property* throughout this paper. The vertex sets associated with nodes of a tree decomposition are called *bags*. We say that two bags are adjacent if the corresponding nodes are adjacent. For an example, refer Figure 1.

▶ **Definition 2.** *The width of a tree decomposition $(T, (X_t)_{t \in V(\mathcal{T})})$ is $max_{t \in V(T)} |X_t| - 1$.*

▶ **Definition 3.** *The treewidth $tw(G)$ of a graph G is the minimum width of all tree decompositions of G.*

The treewidth of a graph G can alternatively be defined in terms of its triangulations. A *chord* of a cycle C in G is an edge whose endpoints lie in the vertex set of C but that is not in the edge set of C [16]. It is easy to see that all cycles of length 3 (i.e., triangles) have no chords. G is *chordal* (or *triangulated*) if every cycle of length at least 4 has a chord. A graph $H = (V(G), E')$ is a *triangulation* of G if H is chordal and G is a subgraph of H, i.e., $E' \supseteq E(G)$. A triangulation is *minimal* if the edge set is inclusion-wise minimal with respect to these conditions. In [10], Bouchitté and Todinca show that there is a minimal triangulation such that a tree decomposition of minimal width can be obtained by constructing a bag for each maximal clique of the triangulation. These bags can be connected to a tree satisfying consistency and the intersections between neighboring bags are minimal separators.

▶ **Definition 4.** *A potential maximal clique (pmc) of a graph G is a vertex set that induces a maximal clique in some minimal triangulation of G.*

This definition, however, is difficult to work with algorithmically. In [10], Bouchitté and Todinca show that potential maximal cliques can also be characterized by local features:

▶ **Definition 5.** *A vertex set $K \subset V(G)$ is cliquish if for every pair of distinct vertices $u, v \in K$ there exists a path from u to v that does not lead through other vertices of K.*

▶ **Lemma 6** ([10], Theorem 3.15)**.** *A vertex set $K \subset V(G)$ is a potential maximal clique if and only if K is cliquish and has no full components associated with it.*

▶ **Definition 7.** *A canonical tree decomposition of a graph G is a tree decomposition of G, where each bag is a potential maximal clique of G and for every two adjacent bags X and Y, $X \cap Y$ is a minimal separator.*

The follwoing two lemmas follow directly from the theory of Bouchitté and Todinca [10] and are also used by Tamaki [14]. The first one summarizes the discussion above.

▶ **Lemma 8.** *For each graph G there is a canonical tree decomposition of width $tw(G)$.*

▶ **Lemma 9.** *Let $\mathcal{T} = (T, (X_i)_{i \in V(T)})$ be a canonical tree decomposition and let $\mathcal{T}' = (T', (Y_j)_{j \in V(T')})$ be any tree decomposition of a graph G, such that for every Y_j there exists X_i with $Y_j \subseteq X_i$. Then for every X_i there exists Y_s with $X_i = Y_j$.*

In other words, if all bags of a tree decomposition are subsets of bags of a canonical tree decomposition, then all bags in the canonical tree decomposition are also part of the other tree decomposition.

We informally argue that Lemma 9 holds as follows: Transforming each bag of a tree decomposition into a clique will result in a triangulation. If the lemma would not be satisfied, the triangulation of the tree decomposition $\mathcal{T}'$ is a strict subset of the triangulation of $\mathcal{T}$.

3 The Algorithm

The algorithm presented by Tamaki [14] decides whether a graph G has a treewidth of at most k, for k ranging from a lower bound on $tw(G)$ to $tw(G)$ (which is, of course, unknown in advance). This is done by trying to construct a canonical tree decomposition of G with width k. The construction is performed by determining all possible candidates for leaves of such a tree decomposition and then iteratively forming larger tree structures by combining two of them along with applying the canonization rules resulting from Lemma 15. This procedure succeeds only in the case of $k = tw(G)$, yielding a canonical tree decomposition of G with optimal treewidth. In the following, it is easier to assume G to be connected, which is guaranteed after the preprocessing techniques discussed in Section 5 (tree decompositions can be computed for each connected component separately).

3.1 Theoretical Foundations of the Algorithm

▶ **Definition 10.** *Let G be a graph and $\mathcal{T} = (T, (X_t)_{t \in V(T)})$ be a rooted canonical tree decomposition of G. For a bag Y_t with $t \in V(T)$, let T_{Y_t} be the subtree of T with root t and all descendants. $\mathcal{T}_Y = (T_Y, (X_t)_{t \in V(T_Y)})$ is called the partial tree decomposition of $\mathcal{T}$ rooted at Y.*

We abbreviate the term *partial tree decomposition* to *ptd*. The algorithm will construct subtrees of tree decompositions, but we will not show that these are necessarily canonical.

▶ **Definition 11.** *For a ptd $\mathcal{T} = (T, (X_t)_{t \in V(T)})$ with root bag X_r we define*

- *$bag(\mathcal{T}) = X_r$,*
- *$V(\mathcal{T}) = \bigcup_{t \in V(T)} X_t$,*
- *$outlet(\mathcal{T}) := N(V(G) \setminus V(\mathcal{T}))$,*
- *$inlet(\mathcal{T}) := V(\mathcal{T}) \setminus outlet(\mathcal{T})$,*
- *a child of $\mathcal{T}$ as the ptd induced by a child node of $\mathcal{T}$ and its descendants, and*
- *$children(\mathcal{T})$ as the set of all children of $\mathcal{T}$.*

Note that $outlet(\mathcal{T}) \subseteq bag(\mathcal{T})$. This is easy to see: the vertices in $outlet(\mathcal{T})$ are exactly those vertices having neighbors that do not lie in $V(\mathcal{T})$. Hence, the edge between such a vertex and its neighbor has to be covered by a bag outside of $\mathcal{T}$. By the consistency property of tree decompositions, however, all bags containing a given vertex have to be connected, implying that all vertices in $outlet(\mathcal{T})$ are also in $bag(\mathcal{T})$.

The second type of structure created by the algorithm is called a *partial tree decomposition with unfinished root* (or *ptdur*). Ptdurs generalize ptds and their role is to gather several ptds under one root. We will later see how they can be extended into ptds using the canonization rules obtained from Lemma 15.

▶ **Definition 12.** *Let $\mathcal{T}_1, ..., \mathcal{T}_n$ be ptds. A partial tree decomposition with unfinished root is a rooted tree with its root labeled $\bigcup_{i \in I} outlet(\mathcal{T}_i)$ and the $\mathcal{T}_i$ as its children.*

We extend the notation from Definition 11 to ptdurs. Note that we do not require the root of a ptdur to be a potential maximal clique, so ptdurs are not necessarily pdts as well. So far, ptdurs also do not necessarily satisfy the consistency property. We fix the latter by immediately discarding any ptdur that is built by the algorithm if it is not possibly usable:

▶ **Definition 13.** *A ptdur $\mathcal{T}$ is possibly usable if for every pair of distinct children $\mathcal{T}_i, \mathcal{T}_j$*

- *$inlet(\mathcal{T}_i) \cap inlet(\mathcal{T}_j) = \emptyset$, and*
- *$V(\mathcal{T}_i) \cap V(\mathcal{T}_j) \subseteq outlet(\mathcal{T}_i) \cap outlet(\mathcal{T}_j)$.*

The following lemma gives all candidates for leaves in a canonical tree decomposition.

▶ **Lemma 14.** *Let $\mathcal{T} = (T, (X_t)_{t \in V(T)})$ be a canonical tree decomposition of a graph G. Let X_l be a bag associated with a leaf in $\mathcal{T}$. Then $X_l = N[v]$ for a vertex $v \in V(G)$.*

Proof. If $\mathcal{T}$'s only bag is X_l, then X_l is a clique in G. This is argued as follows: Suppose X_l is not a clique, then there exist $u, v \in X_l$ such that (u, v) is not an edge in G. In that case the tree decomposition consisting of the bags $X_l \setminus \{v\}$ and $X_l \setminus \{u\}$ is another tree decomposition of G whose bags are proper subsets of X_l. This contradicts Lemma 9.

Otherwise, $\mathcal{T}$ has at least two bags and thus a parent of the leaf associated with X_l exists. Denote its bag by X_p. Since $\mathcal{T}$ is canonical and no bag is a subset of another bag, there exists $v \in X_l \setminus X_p$. Because of the consistency property, v does not lie in any other bag

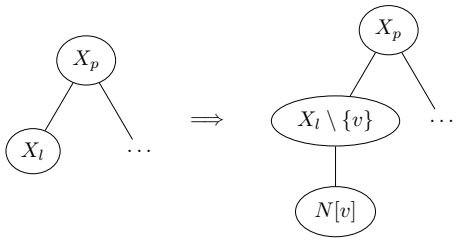


Figure 2 We illustrate the proof of Lemma 14. Replacing the node with the bag X_l by two nodes with the bags $X_l \setminus \{v\}$ and $N[v]$ in $\mathcal{T}$ yields an alternative tree decomposition of G, the existence of which leads to a contradiction.

of $\mathcal{T}$ and has therefore no neighbors outside of X_l. Hence, $N[v] \subseteq X_l$. Now suppose that $N[v] \subsetneq X_l$, i.e., there is $u \in X_l \setminus N[v]$. Then, because v and u are not adjacent in G, G has an alternative tree decomposition where the node associated with X_l is replaced by two nodes with the bags $N[v]$ and $X_l \setminus \{v\}$. Both of these bags, however, are subsets of X_l, so it follows with Lemma 9 that one of them is actually equal to X_l. This is impossible because $N[v]$ does not contain u, whereas $X_l \setminus \{v\}$ does not contain v. Thus, $X_l = N[v]$. ◀

A figure illustrating the proof is given in Figure 2. The following lemma generalizes this result to all bags of a canonical tree decomposition.

▶ **Lemma 15.** *Let $\mathcal{T}$ be a canonical tree decomposition of a graph G. Let $\mathcal{T}'$ be a ptd of $\mathcal{T}$ and let $O = \bigcup_{\mathcal{T}'_c \in children(\mathcal{T}')} outlet(\mathcal{T}'_c)$ and $W = \bigcup_{\mathcal{T}'_c \in children(\mathcal{T}')} V(\mathcal{T}'_c)$ denote the union of the outlets or vertices of its children, respectively. Then, for the root bag X of $\mathcal{T}'$, one of the following conditions holds:*

1. *$X = O$,*
2. *$X = N[v]$ for a vertex $v \in V \setminus W$, such that $O \subseteq N[v]$, or*
3. *$X = O \cup (N(v) \setminus W)$ for a vertex $v \in O$.*

Proof. Throughout this proof, we refer to the example in Figure 6 in the Appendix.

We established earlier in this section that $outlet(\mathcal{T}_i^X) \subseteq bag(\mathcal{T}_i^X)$ for every $i \in I$ because all vertices in the outlet have to lie in at least one other bag of $\mathcal{T}$ outside of $\mathcal{T}_i^X$. Since by the consistency property all bags containing a given vertex have to be connected and the only other incident bag is X, that vertex must also be in X. By repeating this argument for all vertices in O, we get $O \subseteq X$.

If $O = X$, case 1 applies (see Figure 6b).

Otherwise, $O \subsetneq X$. We distinguish between two cases:

▶ **Case 1.** *There exists a vertex $v \in X \setminus (O \cup outlet(\mathcal{T}'))$.*

The proof for this case is similar to the proof of Lemma 14 above. As v is neither contained in the intersection of X with the bag of the parent of X, nor in the intersection of X with any bag of one of its children, it follows that X is the only bag containing v by the consistency property. Consequently, all edges incident to v have to be covered by X, i.e., $N[v] \subseteq X$. Now suppose that $N[v] \subsetneq X$, i.e., there exists $u \in X \setminus N[v]$ (see Figure 6c). Then we can replace the node associated with X by two nodes associated with the bags $X \setminus \{v\}$ and $N[v]$ to construct an alternative tree decomposition of G. The newly introduced bags are subsets of X and by Lemma 9 one of them has to be equal to X. Because $X \setminus \{v\}$ does not contain v and $N[v]$ does not contain u, however, we reach a contradiction. Hence, in this case, X is of type 2 as claimed in the lemma.

▶ **Case 2.** *The set $X \setminus (O \cup outlet(\mathcal{T}'))$ is empty.*

In this case it follows that every vertex in $X \setminus O$ lies in $outlet(\mathcal{T}')$ and therefore also in the bag associated with the parent of $\mathcal{T}'$. At least one vertex $v \in O$ cannot lie in the parent's bag because otherwise X would be a subset of it. Hence, all of v's neighbors have to be covered somewhere within $\mathcal{T}'$. The vertices in $N(v)$ that do not appear in W, therefore, have to be covered by X. Thus, $N(v) \setminus W \subset X$.

It remains to show that there is only one such vertex v. Suppose this is false, i.e., there exist two vertices $u, v \in O \setminus outlet(\mathcal{T}')$ such that neither $N(u) \setminus W \subseteq N(v) \setminus W$ nor $N(v) \setminus W \subseteq N(u) \setminus W$ (see Figure 6f). Then we can replace the node associated with X by two nodes associated with the bags $X \setminus \{v\}$ and $Y := X \setminus ((N(u) \setminus I) \setminus (N(v) \setminus I))$ to construct an alternative tree decomposition of G. The newly introduced bags are once again subsets of X, so by Lemma 9, one of them has to be equal to X. However, $X \setminus \{v\}$ does not contain v and u has a neighbor that is contained in X but not in Y. Therefore, only one such vertex v can exist and X is of type 3 as claimed in the lemma. ◀

These two lemmas are the basis for the algorithm. By enumerating all possible ptdurs and constructing all possible ptds with width at most $tw(G)$, a tree decomposition of G is constructed eventually.

3.2 The Basic Algorithm

Algorithm 1 is a pseudocode version of the basic algorithm. In it, P and U are sets containing the ptds and ptdurs constructed by the algorithm, respectively. The `foreach` loops in lines 5 and 9 iterate over the mutated sets, i.e., also over elements that are added during the execution of the loops.

The algorithm constructs all ptds as stated in Lemmas 14 and 15 (lines 17–27) and all ptdurs by either setting the outlet of a ptd as the root bag and adding no further child (lines 6–7) or adding a new ptd as an additional child to a ptdur constructed before (lines 12–16). Notice that lines 9–10 ensure that the ptdur without an additional child is considered when constructing ptds from ptdurs in lines 17–27.

Intuitively, the algorithm constructs all ptds of height 0 by means of Lemma 14. Then we construct further ptds by constructing ptdurs having subsets of ptds as children: For each ptd $\mathcal{T}$ constructed, we construct a ptdur having $\mathcal{T}$ as its only child. Furthermore, we add $\mathcal{T}$ as an additional child to each ptdur already constructed. We remove all ptdurs that are not possibly usable anymore. Furthermore, we try to make a ptd out of each ptdur via Lemma 15. A formal proof of the correctness is as follows.

▶ **Theorem 16.** *Given G and k, Algorithm 1 returns "YES" if and only if $tw(G) \leq k$.*

Proof. First, we notice that each element $\mathcal{T}$ added to P is a tree decomposition of the graph induced by $V(T)$ and with a width of at most k. Further, all vercies of $V(\mathcal{T})$ adjacent to a node in $V(G) \setminus V(T)$ are in outlet($\mathcal{T}$). Hence, if the algorithm answers "YES", it has found a tree decomposition of G with a width of k.

It remains to show that we find a tree decomposition of width k if there exists one. Hence, we need to show that, at line 28, P contains exactly those ptds of G that have a width smaller or equal to k and that line 28 is indeed reached.

By Lemma 14, all candidates for leaves are added to P in lines 1–4. We now prove that, by the time the algorithm reaches line 28, U contains all possibly usable ptdurs with a width of at most k (and only those). Because lines 17–18, 19–22, and 23–27 correspond to step 2 in Lemma 15, it then follows that P contains all ptds with width at most k when line 28 is reached.

Algorithm 1 Treewidth.

Input : Graph G, positive integer k
Output : "YES" if $tw(G) \leq k$, otherwise "NO"

1 **foreach** $v \in V(G)$ **do**
2 **if** $|N[v]| \leq k+1$ **and** $N[v]$ *is pmc* **then**
3 $p_0 :=$ ptd with bag $N[v]$ as only bag
4 add p_0 to P
5 **foreach** $\mathcal{T} \in P$ **do**
6 $\widetilde{\mathcal{T}} :=$ ptdur with $outlet(\mathcal{T})$ as root bag and $\mathcal{T}$ as only child
7 add $\widetilde{\mathcal{T}}$ to U
8 **foreach** $\mathcal{T}' \in U$ **do**
9 **if** $\widetilde{\mathcal{T}} = \mathcal{T}'$ **then**
10 $\widehat{\mathcal{T}} := \mathcal{T}'$
11 **else**
12 $\widehat{\mathcal{T}} :=$ ptdur obtained from $\mathcal{T}'$ by adding $\mathcal{T}$ as a child and adding $outlet(\mathcal{T})$ to the root bag
13 **if** $\widehat{\mathcal{T}}$ *is possibly usable* **and** $bag(\widehat{\mathcal{T}})$ *is cliquish* **and** $|bag(\widehat{\mathcal{T}})| \leq k+1$ **then**
14 add $\widehat{\mathcal{T}}$ to U
15 **else**
16 **continue**
17 **if** $bag(\widehat{\mathcal{T}})$ *is pmc* **then**
18 add $\widehat{\mathcal{T}}$ to P
19 **foreach** $v \in V(G) \setminus V(\widehat{\mathcal{T}})$ **do**
20 **if** $bag(\widehat{\mathcal{T}}) \subseteq N[v]$ **and** $N[v]$ *is pmc* **and** $|N[v]| \leq k+1$ **then**
21 $p_2 := \widehat{\mathcal{T}}$, where the root bag is replaced by $N[v]$
22 add p_2 to P
23 **foreach** $v \in bag(\widehat{\mathcal{T}})$ **do**
24 $B := bag(\widehat{\mathcal{T}}) \cup \left(N(v) \setminus inlet(\widehat{\mathcal{T}})\right)$
25 **if** B *is pmc* **and** $|B| \leq k+1$ **then**
26 $p_3 := \widehat{\mathcal{T}}$, where B is added to the root bag
27 add p_3 to P
28 **if** P *contains a ptd that covers* $V(G)$ **then**
29 **return** "YES"
30 **else**
31 **return** "NO"

Denote by $p_1, p_2, \ldots$ the ptds constructed by the algorithm in the order that P is iterated over.

▷ Claim 17. For each n, by the time the iteration over p_n (line 5) finishes, U contains all possibly usable ptdurs with a width of at most k, whose children are subsets of $\{p_1, \ldots, p_n\}$.

Proof. We prove the claim by induction.

Base case $n = 1$:

The ptdur with the only child p_1 has the same width as p_1 and is added to U at line 7.

Inductive hypothesis:

Suppose the claim holds for all values of n up to some $l, l > 1$.

Inductive step:

Let $n = l + 1$. By the inductive hypothesis, U already contains all possibly usable ptdurs with a width of at most k whose children are subsets of $\{p_1, \ldots, p_{n-1}\}$. Notice that a ptdur that is not possibly usable does not become possibly usable by adding another child to it. Therefore, the only possibly usable ptdurs left to be constructed are of two types: (1) those that are already in U but have p_n as an additional child, and (2) the ptdur whose only child is p_n. The latter is added to U at line 7, the rest is added at line 14. ◁

It remains to show that the algorithm terminates. Since P and U are sets and therefore do not contain duplicates, it is sufficient to argue that there exist only finitely many non equivalent ptds and ptdurs of G. As ptds $\mathcal{T}$ and $\mathcal{T}'$ are equivalent if $V(\tau) = V(\tau')$ there are at most 2^n ptds. Similarly, we have at most one ptdur with a certain vertex set as inlet and a certain vertex set as root bag and hence at most 2^{2n} in total. ◀

4 Reducing the number of PTDs and PTDURs

In the following sections we describe techniques for reducing the number of ptds and ptdurs considered. Some of these, namely those mentioned in Sections 4.1 to 4.4 and the technique regarding ptdurs with root bags of size $k + 1$ in Section 4.5, are also used by Tamaki's implementation. We review them here for the sake of completeness and to show how they can be utilized in the reinterpreted algorithm.

4.1 Rejecting PTDs with Non-Canonical Root Bags

If a vertex set is a potential maximal clique, it must be cliquish. Notice, however, that all of its subsets are cliquish as well. By contraposition, adding more vertices to a non-cliquish vertex set will not make it cliquish. We can use this fact to reject ptdurs whose roots are not cliquish, because none of the ptd candidates built from them can be actual ptds. Hence, we can reject those ptdurs immediately at line 12.

By Definition 7, the intersection between two adjacent bags in a ptd $\mathcal{T}$ is a minimal separator. For $\mathcal{T}$ to be a child of another ptd $\mathcal{T}'$, it is necessary that the intersection between $bag(\mathcal{T})$ and $bag(\mathcal{T}')$, namely $outlet(\mathcal{T})$, is a minimal separator. Consequently, all ptds whose outlet is not a minimal separator can be rejected.

4.2 Equivalence of PTDs and PTDURs

Let $\mathcal{T}$ be a canonical tree decomposition of G. For two adjacent bags X and Y of $\mathcal{T}$, denote by $\mathcal{T}^{X,Y}$ the tree that is obtained by splitting $\mathcal{T}$ at the edge between the nodes labeled X and Y and discarding the part containing the node labeled with Y. Notice that the $\mathcal{T}^{X,Y}$ can be replaced in $\mathcal{T}$ by any other tree $\mathcal{T}'$ with $V(\mathcal{T}') = V(\mathcal{T}^{X,Y})$. In this sense, $\mathcal{T}^{X,Y}$ and $\mathcal{T}'$ are equivalent. Consequently, we can adjust P to accept a new ptd only if no equivalent ptd is already stored within.

To apply this principle also to ptdur's, we need to consider the size of their root bags as well. This is because a small root bag may be able to be extended to a pmc in more ways than a large root bag, when only bags of size at most $k + 1$ are allowed. See Figure 3 for an

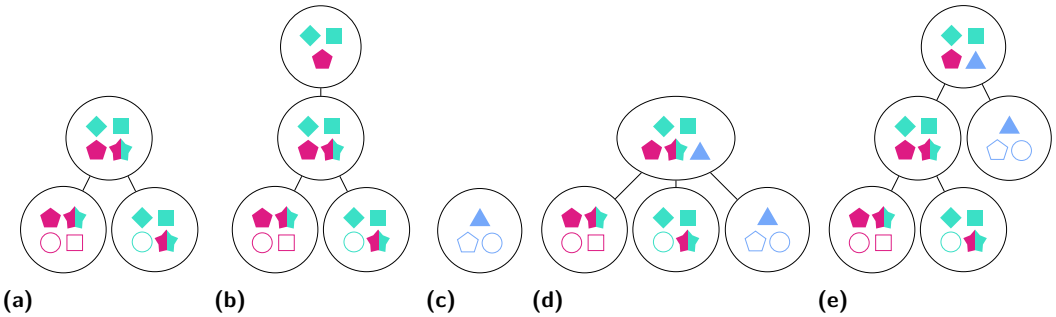


Figure 3 We illustrate the equivalence of ptds. In this figure, each unique vertex is represented as a unique combination of a geometric shape and a color. Vertices in the outlet of a leaf are filled, while vertices in the inlet of a leaf are not. The ptdurs (a) and (b) cover the same vertex set, but are not equally useful. This becomes apparent when adding the ptd (c) as a child to them: the resulting ptdurs (d) and (e) have different widths. If the root bag of (e) is a pmc, then (e) is a ptd of width 3. Hence, during the iteration where $k = 3$, it would be wrongly skipped.

illustration of where this is relevant. Therefore, if we try to add a ptdur $\mathcal{T}$ with root bag X to U, we first test whether an equivalent ptdur, whose root bag is a subset of X, exists already in U. If so, $\mathcal{T}$ is discarded. Otherwise, if a ptdur, whose root bag is a superset of X, exists in U, we replace that ptdur by $\mathcal{T}$. If none of these cases apply, $\mathcal{T}$ is added to U as usual.

4.3 Choosing a Unique Root

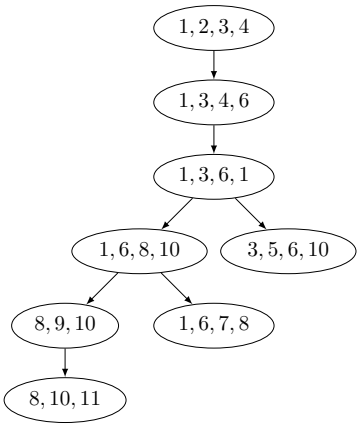
In order to define ptds in Section 3.1, we required the tree decomposition to be rooted. So far, we allow any bag to be the root. This means that we construct each tree decomposition not only once but once for every possible root (which can be any node). We overcome this problem by directing the edges of tree decompositions away from the root and rejecting ptds whose edges would not be consistent with that. We want to define a property called *incoming* for ptds that satisfies the following conditions:

1. for every two adjacent bags X and Y in $\mathcal{T}$, exactly one of $\mathcal{T}^{X,Y}$ and $\mathcal{T}^{Y,X}$ is incoming,
2. for every ptd with root bag X and its neighbors $Y_1, ..., Y_k$ in $\mathcal{T}$, at most one of the $\mathcal{T}^{X,Y_i}, 1 \leq i \leq k$ is incoming, and
3. given a ptd, we can determine whether it is incoming using only information contained within it.

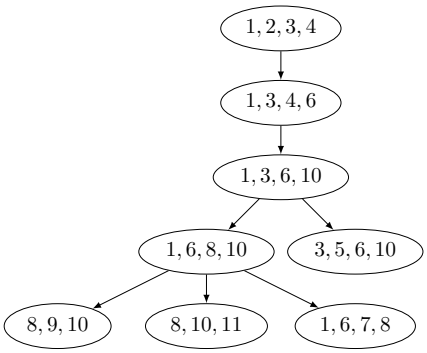
Condition 1 guarantees that each edge has a unique direction. Because we want the edges to be directed away from the root, every bag can have at most one incoming neighbor (condition 2), which is its parent. Condition 3 allows us to use this property in the algorithm. Only if a ptd is not incoming will we add it to P.

Assume that a total ordering $<$ on the vertices of G is given. For a vertex set $U \subseteq V(G)$, we define $min(U)$ as its smallest element under $<$. If U is empty, then we consider $min(U)$ to be smaller than any vertex. Then the following definition of incoming satisfies the conditions given above.

▶ **Definition 18.** *A ptd $\mathcal{T}$ is incoming if $min(inlet(\mathcal{T})) < min(V \setminus V(\mathcal{T}))$.*



(a) The tree decomposition from Figure 1b, where all ptds are required to be not incoming, thereby choosing $N[2] = \{1, 2, 3, 4\}$ as the unique root.



(b) The same tree decomposition, where all ptds are normalized. The bag $\{8, 10, 11\}$ has been attached to its former grandparent $\{1, 6, 8, 10\}$ instead.

Figure 4 The tree decomposition shown in Figure 1b has leaves $N[2], N[5], N[7]$, and $N[11]$, so its unique root is chosen to be $N[2]$ (see Figure 4a). Notice, however, that $N[9] = \{8, 9, 10\}$ is also a candidate for a leaf, but is actually an inner node here. This means that it could have been chosen as the root under another ordering of vertices. While this is not a conflict when choosing a unique root, it illustrates another way in which some ptds are redundant. Consider the ptd consisting of only the bag $\{8, 10, 11\}$. Its outlet is $\{8, 10\}$, which is contained in its grandparent $\{1, 6, 8, 10\}$. It is therefore possible to attach the bag to its grandparent instead (see Figure 4b). We call a tree where this is not the case for any bag *normalized*.

It is not hard to see that the three properties mentioned above hold:

1. As $inlet(\tau^{X,Y}) = V \setminus V(\mathcal{T}^{Y,X})$ and vice versa.
2. The sets $inlet(\tau^{X,Y_i})$ are pairwise disjoint. Let v be the smallest vertex in the union of these sets and ℓ^* be such that $v \in inlet(\tau^{X,Y_{\ell^*}})$. For each $\ell \neq \ell^*$, we have $v \in V \setminus V(\mathcal{T}^{X,Y_\ell})$ and each node in $inlet(\tau^{X,Y_\ell})$ is larger than v. Hence T^{X,Y_ℓ} is not incoming.
3. As the definition uses only $inlet(\tau)$ and $V(\tau)$.

Since $\mathcal{T}$ is equivalent to ptds with the same vertex set or, equivalently, the same inlet, we also say that $inlet(\mathcal{T})$ is incoming if a ptd $\mathcal{T}$ is incoming. For every tree decomposition, this procedure will choose the leaf $N[v]$ as the root, for which v is smallest out of all the leaves contained in that tree decomposition.

An example is shown in Figure 4a.

4.4 Normalization of PTDs

Notice that if the outlet of a ptd $\mathcal{T}'$ used in the construction of the ptd $\mathcal{T}$ is contained in several bags of $\mathcal{T}$, we can attach $\mathcal{T}'$ to any one of those bags. To avoid enumerating all these possibilities, we normalize the ptd $\mathcal{T}$ by enforcing that $\mathcal{T}'$ is attached as close to the root of $\mathcal{T}$ as possible. This is captured in the following definition.

▶ **Definition 19.** *Let $p(t)$ denote the parent of a node t within a rooted tree.*

A ptd $\mathcal{T} = (T, (X_t)_{t \in V(T)})$ is normalized if there is no $t \in V(T)$ such that $outlet(T_t^+) \subseteq X_{p(p(t))}$.

This means that we can discard a ptd $\mathcal{T}$ if $outlet(\mathcal{T}_C) \subseteq outlet(\mathcal{T})$ for one of its children $\mathcal{T}_C$.

4.5 Rejecting PTDURs Whose Root Cannot Be Extended To a Potential Maximal Clique

The roots of some of the ptdurs created at line 12 are not pmcs. If no vertices can be added to them so that they finally become pmcs either, they are useless and can be discarded.

First, we consider a ptdur whose root contains $k+1$ vertices, i.e., we can add no more vertices to it. If it is not a pmc, then the ptdur is useless. Next, we consider a ptdur whose root X contains k vertices, i.e., we are allowed to add one more vertex to it. If X is a pmc, we accept the ptdur as usual. If not, we can test if, for any candidate $v \in V(G) \setminus V(\mathcal{T})$, $X \cup \{v\}$ is a pmc. This test may take a lot of processing time, so we narrow down the list of candidates by studying necessary conditions on candidates.

Let C be the component associated with X that contains a candidate v. We can assume that X is cliquish because otherwise it will be rejected at line 13. In order for $X \cup \{v\}$ to be cliquish as well, there has to exist a path between v and x for each $x \in X$ that does not lead through other vertices in X. This implies that C must be a full component because otherwise there would exist $x \in X$ such that $x \notin N(C)$ and hence, every path from v to x would contain another vertex of X.

On the other hand, in order to be a pmc, $X \cup \{v\}$ can have no full components associated with it, which leaves us with two possibilities: either v separates $G[C]$ into at least two components, neither of which is full in G with respect to $X \cup \{v\}$. Or v does not separate $G[C]$, in which case there needs to exist $x \in X$ such that $x \notin N(C \setminus \{v\})$, i.e., v is the only neighbor of x in C.

4.6 PTD Outlets With More Than 2 Associated Components

Let S be a minimal separator of G with associated components $C_1, C_2, ..., C_\ell$ such that $\ell \geq 3$. Without loss of generality, let C_1 be the incoming component, let C_2 be a full (non-incoming) component, and let $\mathcal{T}_2$ be a ptd with $\mathit{inlet}(\mathcal{T}_2) = C_2$. Because C_2 is a full component associated with S, we have $\mathit{outlet}(\mathcal{T}_2) = S$.

▶ **Lemma 20.** *If $\mathcal{T}_2$ is contained in a tree decomposition of G with a width of at most k, then there exist non-incoming ptds $\mathcal{T}_3, ..., \mathcal{T}_\ell$, whose inlets are $C_3, ..., C_\ell$, respectively, and whose widths are at most k*

Proof. Notice that any tree decomposition of G that contains the ptd $\mathcal{T}_2$ is also a tree decomposition of the graph $G\langle S\rangle$. Let $\mathcal{T}$ be such a tree decomposition and let C be any component associated with S. From all bags of $\mathcal{T}$, remove all vertices except those contained in $S \cup C$. The resulting tree decomposition is a tree decomposition of $G\langle S\rangle[S \cup C]$ with a width of at most k. By Lemma 8, that graph also has a canonical tree decomposition of equal or smaller width. Interpreted as a ptd of G, it has inlet C as claimed. ◀

The contrapositive of this lemma states that $\mathcal{T}_2$ is only useful if we can also construct $\mathcal{T}_3, ..., \mathcal{T}_\ell$ (or equivalent ptds). We can use this insight to delay the addition of $\mathcal{T}_2$ to P until ptds covering all non-incoming components associated with S are constructed.

4.7 Using Upper and Lower Bounds

As another approach, we tried to heuristically complete ptds (who cover only part of the graph) to a tree decomposition on the entire graph, using the min-degree and min-fill heuristics. If a heuristic succeeds in finding a tree decomposition of the complete graph with the given width, we can immediately stop the computation and return the result.

Furthermore, we can compute lower bounds for a tree decomposition that contains a given ptd. We complete the bag of the root to a clique and compute a lower bound of the resulting graph. If this lower bound is larger than the given treewidth, we can remove the ptd of our list. To improve the time to compute the lower bound, we could only consider the part of the graph that is not covered by the ptd. A very simple lower bound is the second lowest degree. As in [9], we determine a lower bound by heuristically computing a minor of G and subsequently trying to increase this very simple lower bound.

5 Further Details of the Algorithm

We implemented several preprocessing methods that reduce the size of the graph before entering the main loop. More precisely, we implemented the simplicial vertex rule, almost simplicial vertex rule, buddy rule, and cube rule, all of which can be found in [4]. All these rules are based on criteria guaranteeing that the complete neighborhood of a vertex is contained in single bag in any tree decomposition. Completing the neighborhood to a clique and removing the vertex itself results in a graph with one vertex less such that any tree decomposition still contains the neighborhood in a single bag. Hence, we can compute the tree decomposition of the resulting graph and add a bag containing the vertex removed together with its neighborhood to the tree decomposition and add it at the appropriate position.

Furthermore, we implemented the heuristic to find safe separators of [14]. A separator S for G is called safe if the treewidth of G is the maximum of the treewidth of the graphs $G[C \cup S]\langle S\rangle$ for the components C of $G[V(G) \setminus S]$. Having computed a safe separator, we can compute tree decompositions independently for each subgraph and connect the trees at the bags containing the separator. Tamaki's heuristic tries to construct separators and use a sufficient criterion of a separator S to be safe proven by Bodlaender and Koster [8], namely that for each component C of $G[V(G) \setminus S]$ the graph $G[V(G) \setminus C]$ has S as a labeled minor. Furthermore, we implemented an algorithm to find separators that are cliques or almost cliques as given in [8].

We extend the usage of safe separators as follows: whenever we construct a new ptd, we test whether its outlet is a safe separator with Tamaki's heuristic, unless the separator has already been tested before.

Instead of testing at the end of each iteration whether we have built a ptd covering $V(G)$, we test this condition for every ptd as it is added to P. If so, we return "YES" immediately.

To avoid trying to add every ptd constructed to every ptdur, we implemented (a slight variant of) the block sieve as presented in [14]. In this data structure, the ptdurs are sorted by the size of their root bags. If the size of the outlet of the current ptd plus the size of the root bag of a ptdur is larger than the current value of k, these sets have to coincide on a number of vertices. For each size, the ptdurs are stored in tries, which allow to efficiently iterate over the elements coinciding in a given number of vertices.

6 Experiments

We have compared the performance of our implementation against Tamaki's on the PACE2017 public instances in the *treewidth exact* track. The experiments were conducted in the following environment: Intel Core i5-6600K@3.5GHz CPU, 16GB DDR4 RAM, Windows 10 (64 bit), Java version jre1.8.0_271, .NET version 4.8. The time measured is the CPU time, which includes the time for garbage collection.

For the sake of a fair comparison, we initially enabled only those optimizations that Tamaki also uses in his implementation, namely the following: (1) the rejection of ptdurs with non-cliquish root bags, (2) the rejection of ptds whose outlet is not a minimal separator, (3) the rejection of ptds and ptdurs for whom equivalent ptds or ptdurs have already been built, (4) accepting only non-incoming ptds, (5) rejecting ptds that are not normalized, and (6) rejecting ptdurs whose root bag contains $k+1$ vertices but is not a pmc. Furthermore, we used only the safe separators found by Tamaki's heuristic for this experiment and no further preprocessing.

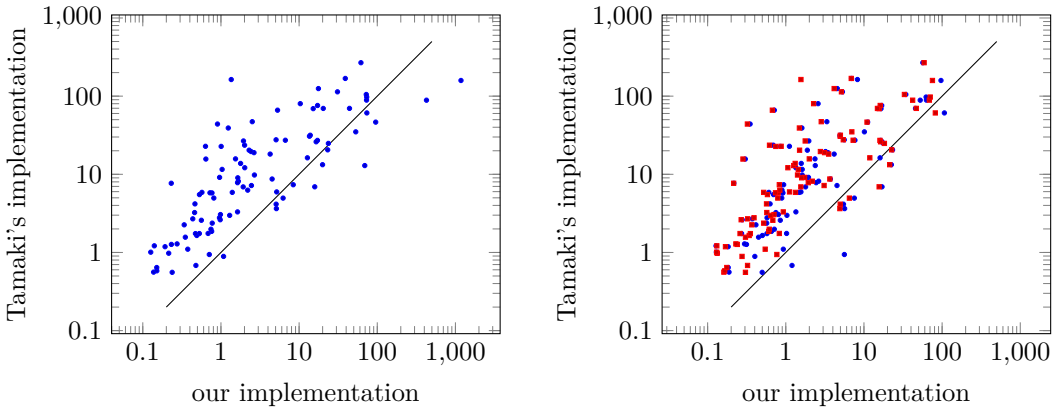
We plot the results in Figure 5a. Every dot corresponds to one of the test instances and its position corresponds to the running times of Tamaki's and our implementation on that instance. A point located above the diagonal line indicates that our implementation is faster. In total, Tamaki's implementation solves the instances in 46 minutes, 24 seconds, whereas we solve the instances in 44 minutes, 7 seconds. Although the difference is small, our implementation beats Tamaki's on 86 instances. This is mainly due to two instances that together make up more than half of the total running time of our implementation. Conversely, the remaining instances are often solved rather quickly.

In Figure 5b, we plot the running time of Tamaki's implementation against ours with the additional techniques of Section 4 and the preprocessing discussed in Section 5 enabled. Our implementation to compute lower bounds is not used as it is not efficient enough. Although the number of ptds and ptdurs are reduced by 5% and 49%, respectively, the running time increased by about 36%. The blue dots represent the running times, where all strategies for ptd and ptdur reduction are enabled, along with the further strategies discussed in Section 5. The strategy to reject ptdurs whose root bag contains k vertices and cannot be extended to a ptd and the search for safe separators during the runtime of the algorithm turned out to be slightly disadvantageous, but only by a small margin. We therefore also plot in red the running time when they are disabled. The total running time with all and only the best strategies enabled, respectively, are 14 minutes, 31 seconds and 13 minutes, 8 seconds, outperforming Tamaki's implementation on 91 and 93 instances, respectively. On larger instances than the ones tested here, however, the techniques excluded from best could actually be beneficial.

We have also evaluated the impact of the strategies to reject ptdurs whose root bag has k vertices and cannot be extended to a pmc and to delay the addition of ptds to P until ptds covering all of its non-incoming components are found. We counted the numbers of ptds and ptdurs enumerated[1] either until we stop as soon as the first (optimal) solution is found or until no further ptds and ptdurs with the current width could be found with various strategies enabled. The later numbers can be interpreted as the size of the search space for each graph. Since we implemented the algorithm such that it returns immediately when a tree decomposition is found, the first give the numbers of the actually enumerated objects which is often far less than that. Our machine ran out of memory while computing the size of the search space for the instance ex003.gr, so it is not included in the results.

The results are summarized in Table 1. The percentages give the average relative reduction of the ptds and ptdurs, respectively, compared to when only the other strategies discussed in this paper are used. Only a small impact can be attributed to the strategy that delays the addition of a ptd to P until ptds covering all its other non-incoming components are found (abbreviated as *>2 comp.*). The strategy to reject ptdurs whose root bag contains k elements

[1] Note that we split graphs at safe separators into smaller graphs, so the graphs we use to count them are actually only subgraphs of the test instances.



(a) A comparison, where both implementations use the same preprocessing and strategies for reducing combinatorial objects.

(b) A comparison, where our implementation uses all preprocessing discussed in Section 5 and all (blue) and only the best (red) strategies for reducing the amount of combinatorial objects.

Figure 5 A comparison of running times in seconds between Tamaki's and our implementation. Every dot represents one instance.

Table 1 The average relative reduction of ptd's and ptdur's when the newly developed strategies for reducing their numbers are employed. A dash signals that that strategy is not able to reduce the number of the respective objects by its conception.

	until first solution found			until all ptd(ur)s are enumerated		
average relative reduction...	pmc k	>2 comp.	all	pmc k	>2 comp.	all
... of ptds	–	4.61%	4.61%	–	0.08%	0.08%
... of ptdurs	34.78%	2.52%	37.29%	44.74%	0.06%	44.78%

but cannot be extended to a pmc (abbreviated as *pmc k*), however, has a huge impact on the number of ptdurs enumerated. Unfortunately, as mentioned above, determining if any candidate vertices can extend such a bag to pmc takes a comparatively long time, which has a slightly negative impact on the total running time over all instances.

Finally, we tried to solve all DIMACS graph coloring instances. For each instance we set a time limit of $30min$. Table 2 in the appendix gives a summary of our finding. For our implementation and the one by Tamaki, we give the running time and the lower bound obtained after the time limit.

7 Conclusion

We gave a description of Tamaki's algorithm to compute the treewidth of a graph that is considerably more accessible than the original formulation. This is archived by the interpretation of the enumerated structures as partial tree decompositions and partial tree decompositions with unfinished root. Furthermore, seeing this interpretation allows us to remove some of the structures from the enumeration. This results in an algorithm that is more efficient in practice.

In future work, we want to derive and implement further techniques to reduce the number of enumerated partial tree decompositions, including better or faster lower bounds. Furthermore, we plan to extend the usage of techniques known for preprocessing within the construction phase.

References

1 Ernst Althaus and Sarah Ziegler. Optimal tree decompositions revisited: A simpler linear-time FPT algorithm. *CoRR*, abs/1912.09144, 2019. arXiv:1912.09144.

2 Ernst Althaus and Sarah Ziegler. Optimal tree decompositions revisited: A simpler linear-time fpt algorithm. In: Gentile, C., Stecca, G., Ventura, P. (eds) Graphs and Combinatorial Optimization: from Theory to Applications (CTW2020 Proceedings), 2020. AIRO Springer Series, vol 5. Springer, 2021.

3 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM JOURNAL OF DISCRETE MATHEMATICS*, 8(2):277–284, 1987.

4 Hans Bodlaender, Arie Koster, Frank Eijkhof, and Linda C. Gaag. Pre-processing for triangulation of probabilistic networks, April 2002.

5 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

6 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

7 Hans L. Bodlaender and Arie M. C. A. Koster. Safe separators for treewidth. *Discret. Math.*, 306(3):337–350, 2006. doi:10.1016/j.disc.2005.12.017.

8 Hans L. Bodlaender and Arie M. C. A. Koster. Safe separators for treewidth. *Discrete Math.*, 306(3):337–350, 2006.

9 Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations II. lower bounds. *Inf. Comput.*, 209(7):1103–1119, 2011. doi:10.1016/j.ic.2011.04.003.

10 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31:212–232, 2001.

11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

12 Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *ACM J. Exp. Algorithmics*, 23, 2018. doi:10.1145/3173045.

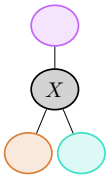
13 Neil Robertson and P.D Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

14 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 68:1–68:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.68.

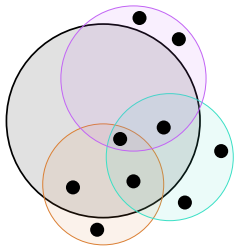
15 Hisao Tamaki. A heuristic use of dynamic programming to upperbound treewidth. *CoRR*, abs/1909.07647, 2019. arXiv:1909.07647.

16 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000.

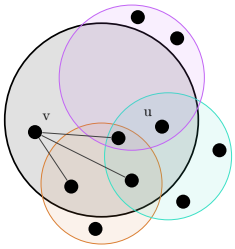
A Appendix



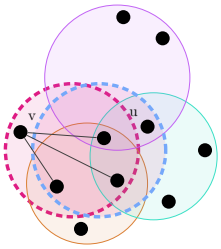
(a) A part of $\mathcal{T}$: the root node of $\mathcal{T}'$ with the bag X, its children, and its parent.



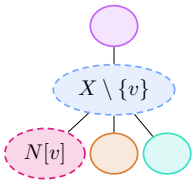
(b) Case $O = X$ of Lemma 15. Shown are vertices of G and their membership of the bags depicted in Figure a.



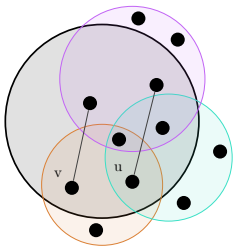
(c) Case 1 of Lemma 15. Assuming that u does not lie in $N[v]$, ...



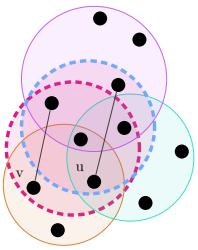
(d) ... one can replace the node associated with X by nodes associated with the bags depicted here as dotted circles, ...



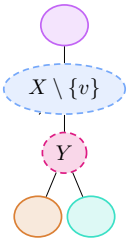
(e) ... thus, the part of $\mathcal{T}$ shown in Figure (a) can be replaced by the one shown here.



(f) Case 2 of Lemma 15. Assuming that both u and v have different neighbors that are not covered in $\mathcal{T}'$s children, ...



(g) ... one can replace the node associated with X by nodes associated with the bags depicted here as dotted circles, ...



(h) ... thus the part of $\mathcal{T}$ shown in Figure (a) can be replaced by the one shown here.

Figure 6 Illustration of the proof of Lemma 15. Figure 6a shows the part of $\mathcal{T}$ induced by X and its adjacent bags, that is, its children (if any) and possibly a parent. The Figures 6b, 6c and 6f show the vertices of G that are covered by the bags in Figure 6 in three different cases. Each bag of the tree decomposition is represented by a circle of the corresponding color and contains all of the vertices within it. Edges are only shown where they are necessary for the argument.

Table 2 The running time in seconds and the lower bound on the treewidth for our implementation and the one of Tamaki with a time limit of 30*min*. It should be noted that a slower computer was used for this comparison than for the calculations before.

	Our impl.		Tamaki	
Name	tw	time	tw	time
anna	12	0,079	12	0,286
david	13	0,044	13	0,234
DSJC1000.1	>0	1800	>187	1800
DSJC1000.5	>0	1800	>739	1800
DSJC1000.9	>0	1800	>986	1800
DSJC125.1	>34	1800	>35	1800
DSJC125.5	>107	1800	108	829,5
DSJC125.9	119	2,137	119	0,069
DSJC250.1	>62	1800	>65	1800
DSJC250.5	>205	1800	>210	1800
DSJC250.9	243	30,26	243	0,993
DSJC500.1	>106	1800	>113	1800
DSJC500.5	>369	1800	>384	1800
DSJC500.9	492	718,7	492	27,05
DSJR500.1	>22	1800	>23	1800
DSJR500.1c	485	319,3	485	5,176
DSJR500.5	>225	1800	246	1187
flat1000_50	>0	1800	>733	1800
flat1000_60	>0	1800	>734	1800
flat1000_76	>0	1800	>735	1800
flat300_20	>228	1800	>244	1800
flat300_26	>230	1800	>246	1800
flat300_28	>230	1800	>246	1800
fpsol2.i.1	66	16,28	66	1256
fpsol2.i.2	31	32,51	>0	1800
fpsol2.i.3	31	31,68	>0	1800
games120	>27	1800	>28	1800
homer	30	1189	>27	1566
huck	10	0,002	10	0,027
inithx.i.1	56	65,80	>0	1800
inithx.i.2	>31	1800	>0	1800
inithx.i.3	31	1751	>0	1800
jean	9	0,002	9	0,008
le450_15a	>73	1800	>45	1052
le450_15b	>75	1800	>44	958,5
le450_15c	>122	1800	>130	1800
le450_15d	>121	1800	>129	1800
le450_25a	>76	1800	>23	581,6
le450_25b	>75	1800	>28	709,3
le450_25c	>112	1800	>105	1800

	Our implementation		Tamaki	
Name	tw	time	tw	time
le450_25d	>112	1800	>109	1800
le450_5a	>62	1800	>58	1800
le450_5b	>63	1800	>59	1800
le450_5c	>94	1800	>98	1800
le450_5d	>93	1800	>97	1800
miles1000	49	1,274	49	0,496
miles1500	77	3,606	77	0,870
miles250	9	0,012	9	0,023
miles500	22	0,467	22	0,149
miles750	36	1,885	36	0,348
mulsol.i.1	50	0,531	50	159,8
mulsol.i.2	32	64,28	32	1424
mulsol.i.3	32	65,80	32	1480
mulsol.i.4	32	67,16	32	1494
mulsol.i.5	31	67,97	31	1572
myciel2	2	0,001	2	0,001
myciel3	5	0,002	5	0,002
myciel4	>0	1800	10	0,004
myciel5	19	23,71	19	0,621
myciel6	>25	1800	>34	1800
myciel7	>42	1616	>33	1213
queen10_10	>66	1800	>68	1800
queen11_11	>73	1800	>76	1800
queen12_12	>80	1800	>83	1800
queen13_13	>86	1800	>90	1800
queen14_14	>92	1800	>97	1800
queen15_15	>98	1800	>103	1800
queen16_16	>104	1800	>105	1800
queen5_5	18	0,054	18	0,007
queen6_6	25	0,103	25	0,024
queen7_7	35	1,550	35	0,484
queen8_12	>63	1800	65	1613
queen8_8	45	18,04	45	9,145
queen9_9	58	1369	58	651,2
school1	>123	1800	>122	1800
school1_nsh	>112	1800	>108	1800
zeroin.i.1	50	8,668	50	39,48
zeroin.i.2	32	1,635	32	185,4
zeroin.i.3	32	1,634	32	186,9