Treewidth Computations I.
Upper Bounds

Hans L. Bodlaender

Arie M. C. A. Koster

Technical Report UU-CS-2008-032
September 2008

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Treewidth Computations I. Upper Bounds*

Hans L. Bodlaender† Arie M. C. A. Koster‡

Abstract

For more and more applications, it is important to be able to compute the treewidth of a given graph and to find tree decompositions of small width reasonably fast.

This paper gives an overview of several upper bound heuristics that have been proposed and tested for the problem of determining the treewidth of a graph and finding tree decompositions. Each of the heuristics produces tree decompositions whose width is not necessarily optimal, but experiments show that many of these come often close to the exact treewidth.

1 Introduction

The notions of *treewidth* and *tree decomposition* have gained their attractiveness partly because many graph and network problems that are intractable (e.g., NP-hard) on arbitrary graphs become more efficiently solvable (e.g., with a linear time algorithm) when the treewidth of the input graphs is bounded by a constant. Such algorithms have been found for many combinatorial problems (see e.g., [3, 6, 46, 60, 63]), and also have been employed for problems from computational biology (see e.g., [64, 65]), constraint satisfaction (see e.g., [25, 36, 46]), and probabilistic networks (see [49]).

Many of the linear or polynomial time algorithms for problems on graphs with small treewidth have the following form. First, a tree decomposition of the graph with small width is found. Then, this tree decomposition is used in a dynamic programming algorithm to solve the original problem. In case of a theoretical investigation where we are not interested in the constant factor hidden in the O-notation, one can use for the first step

*The research was partly carried out during a visit by the second author to Utrecht University with support from the project *Treewidth and Combinatorial Optimization* with a grant from the Netherlands Organization for Scientific Research NWO.

†Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. E-mail: hansb@cs.uu.nl

‡University of Warwick, Centre for Discrete Mathematics and its Applications (DIMAP), Coventry CV4 7AL, United Kingdom. E-mail: Arie.Koster@wbs.ac.uk

Keywords: Treewidth, Upper bounds, Heuristics, Approximation Algorithms, Graph algorithms
Mathematics Subject Classification (2000): 05C85, 68Q25, 68R10

the algorithm from [15]: for each fixed k, there is a linear time algorithm that either tells that the treewidth of a given input graph G is larger than k, or finds a tree decomposition of width at most k. In practice however, the algorithm from [15] is not useful due to the huge constant factor. This was also shown by Röhrig [55] in a experimental evaluation of the algorithm of [15]. Thus, there is a need for *practical* algorithms that find tree decompositions of given graphs of small width.

We aim to address this issue in a series of three overview papers, reviewing the developments of the last decade and complementing some minor results. In this first paper of the series, we look at upper bound heuristics and approximation algorithms, i.e., algorithms, that given a graph G, find a tree decomposition of G whose width is possibly not optimal, but hopefully close to optimal. The paper is accompanied by a website for experiments with some of the algorithms presented [44].

In later papers in this series, we plan to address algorithms that give lower bounds to the treewidth of input graphs [19], exact algorithms, and preprocessing methods [20].

This paper is organised as follows. In Section 2, we give several of the necessary definitions, and some useful graph theoretic results. In particular, we look at some different characterisations of treewidth, which will be of use for different types of heuristics. One such characterisation is with help of elimination orderings, and heuristics based on this notion are discussed in Section 3. Most other heuristics for treewidth appear to be relying in some way on the notion of *separator*. These are discussed in Section 4. Sometimes, when we have found a tree decomposition, it can be improved with help by finding a minimal triangulation inside the triangulation corresponding to the tree decomposition; see Section 5. We report on an experimental study to evaluate some of the heuristics in Section 6. Some final conclusions are given in Section 7.

2 Preliminaries

In this section, we give some definitions and some useful graph theoretic results.

All graphs we consider in this paper are undirected and simple, i.e., without parallel edges or self-loops. A graph is denoted $G = (V, E)$ with V the set of vertices and E the set of edges. Unless stated otherwise, $n = |V|$ denotes the number of vertices in the considered graph. The degree of a vertex $v \in V$ in graph G is denoted $d_G(v)$ or $d(v)$ if the graph is clear from the context.

The set of neighbours of v in graph $G = (V, E)$ is denoted by $N_G(v) = \{w \in V \mid \{v, w\} \in E\}$. The set of neighbours of v and v itself is denoted $N_G[v] = N_G(v) \cup \{v\}$.

We assume the reader to be familiar with notions like cycle, clique, maximal clique, connected component. The subgraph of $G = (V, E)$ *induced* by vertex set $W \subseteq V$ is denoted by $G[W] = (W, \{\{v, w\} \in E \mid v, w \in W\})$.

A set of vertices $S \subseteq V$ is a *separator* (or *separating vertex set*) in a graph $G = (V, E)$ if $G[V - S]$ has more than one connected component. A *minimum separator* is a separator of minimum size. A separator S in $G = (V, E)$ is an s-t-separator for vertices $s, t \in V$, if s and t belong to different connected components of $G[V - S]$. An s-t-separator S is a *minimal* *s-t-separator*, if it does not contain another s-t-separator as a proper subset. A separator S is a *minimal* separator, if there are s, $t \in V$, such that S is a minimal s-t-separator. A separator S is an *inclusion minimal separator*, if it does not contain another separator S' in G as a proper subset.

The notions of treewidth and tree decomposition were introduced by Robertson and Seymour [53] in their work on graph minors.

Definition 1 *A* tree decomposition *of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, with $\{X_i \mid i \in I\}$ a family of subsets of V and T a tree, such that*

- $\bigcup_{i \in I} X_i = V$,
- *for all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$, and*
- *for all $v \in V$, the set $I_v = \{i \in I \mid v \in X_i\}$ forms a connected subtree of T.*

The width *of tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph G, $tw(G)$, is the minimum width among all tree decompositions of G.*

The third condition of tree decomposition can be replaced by the following equivalent condition:

- For all $i_0, i_1, i_2 \in I$: if i_1 is on the path from i_0 to i_2 in T, then $X_{i_0} \cap X_{i_2} \subseteq X_{i_1}$.

An example of a graph with a tree decomposition is given in Figure 1.

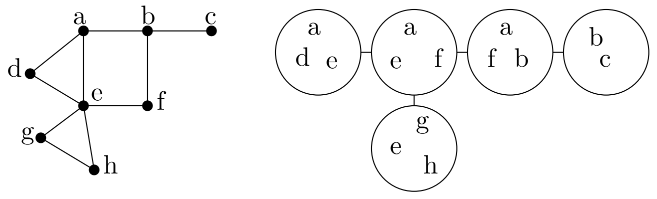


Figure 1: A graph with a tree decomposition

We start with a simple lemma, which is a restatement of the Helly properties for subtrees of a tree, see [23].

Lemma 2 (See [23]) *Let $W \subseteq V$ be a clique in $G = (V, E)$, and $(\{X_i \mid i \in I\}, T = (I, F))$ be a tree decomposition of G. Then there is an $i \in I$ with $W \subseteq X_i$.*

There are several equivalent definitions to the notion of treewidth, some proposed slightly earlier, e.g., the notion of partial k-tree (see [3]). An overview of several such notions can be found in [16]. One alternative characterisation is by the use of elimination orderings. On this notion, some heuristics are based. We introduce the characterisation with help with equivalent characterisations of triangulated graphs, also known as chordal graphs.

Definition 3 *(i). A graph $G = (V, E)$ is* triangulated*, if every cycle in G with length at least four has a chord, i.e., an edge connecting two non-successive vertices in the cycle.*

(ii). An elimination ordering *of a graph $G = (V, E)$ is a bijection $f : V \to \{1, 2, \ldots, n\}$. An elimination ordering f is* perfect*, if for all $v \in V$, the set of its higher numbered neighbours $\{w \mid \{v, w\} \in E \ \wedge \ f(w) > f(v)\}$ forms a clique.*

(iii). A graph $G = (V, E)$ is the intersection graph of subtrees of a tree*, if and only if there is a tree $T = (I, F)$, and for each $v \in V$ a subtree of T, $T_v = (I_v, F_v)$, such that for all $v, w \in V$, $v \neq w$, we have that $\{v, w\} \in E$, if and only if T_v and T_w have at least one vertex in common, i.e., $I_v \cap I_w \neq \emptyset$.*

It is known for over thirty years that triangulated graphs can be alternatively characterised by perfect elimination orderings or as intersection graphs of subtrees of a tree, see [32, 56] or [35, Chapter 4].

Theorem 4 *Let $G = (V, E)$ be a graph. The following are equivalent.*

(i). G is triangulated.

(ii). G has a perfect elimination ordering.

(iii). G is the intersection graph of subtrees of a tree.

(iv). There is a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F)$ of G, such that for each $i \in I$, X_i is a clique in G.

Proof: For equivalence of (i), (ii), and (iii), see [32, 56] or [35, Chapter 4].

(iii) $\to$ (iv): Suppose we have intersection model tree $T = (I, F)$ with for all $v \in V$, subtree T_v with for all $v, w \in V$, $v \neq w$, $\{v, w\} \in E \Leftrightarrow I_v \cap I_w \neq \emptyset$. Now one can easily verify that $(\{X_i \mid i \in I\}, T = (I, F))$ with $X_i = \{v \in V \mid i \in I_v\}$ is a tree decomposition of G, and for all $i \in I$, X_i is a clique.

(iv) $\to$ (iii): Let $(\{X_i \mid i \in I\}, T = (I, F))$ be a tree decomposition of G with for each $i \in I$, X_i is a clique in G. Let for all $v \in V$, $I_v = \{i \in I \mid v \in X_i\}$. By the definition of tree decomposition I_v induces a subtree of T, which we call T_v. Now, the collection of T_v's form the intersection model. □

Definition 5 *A graph $H = (V_H, E_H)$ is a* triangulation *of a graph $G = (V_G, E_G)$, if H is a triangulated graph that is obtained by adding zero or more edges to G ($V_G = V_H$, $E_G \subseteq E_H$). A triangulation $H = (V, E_H)$ is a* minimal triangulation *of $G = (V, E_G)$ if there is no triangulation of G that is a proper subgraph of H, i.e., if there is no set of edges F such that (V, F) is a triangulation of G with $F \subseteq E_H$, $F \neq E_H$.*

The equivalent notions for triangulated graphs can now be translated to equivalent notions for treewidth.

We first give a mechanism that adds edges to a graph to make it triangulated, using an elimination ordering. Consider Algorithm 1. Fill-in(G,π) yields a graph H. One can easily observe that π is a perfect elimination ordering of H; in fact, we added the minimum set of edges to G such that π is a perfect elimination ordering of π. Call H the *fill-in graph* of G with respect to elimination ordering π. As the fill-in graph H has a perfect elimination ordering, it is a triangulation of G.

Algorithm 1 FILL-IN(GRAPH G, ELIMINATION ORDERING π)

```
$H = G$;
for $i = 1$ to $n$ do
  Let $v = \pi^{-1}(i)$ be the $i$th vertex in ordering $\pi$.
  for each pair of neighbours $w$, $x$ of $v$ with $w \neq x$, $\pi(w) > \pi(v)$, $\pi(x) > \pi(v)$ do
    if $w$ and $x$ not adjacent in $H$ then
      add $\{w, x\}$ to $H$.
    end if
  end for
end for
return $H$
```

We now come to the following well known alternative characterisations of the notion of treewidth.

Theorem 6 *Let $G = (V, E)$ be a graph, and let $k \leq n$ be a non-negative integer. The following are equivalent.*

(i). G has treewidth at most k.

(ii). G has a triangulation H such that the maximum size of a clique in H is at most $k+1$.

(iii). There is an elimination ordering π, such that the maximum size of a clique of the fill-in graph of G with respect to π is at most $k+1$.

(iv). There is an elimination ordering π, such that no vertex $v \in V$ has more than k neighbours with a higher number in π in the fill-in graph of G with respect to π.

Proof: (i) $\Rightarrow$ (ii): Suppose $(\{X_i \mid i \in I\}, T = (I, F))$ is a tree decomposition of G of width at most k. Let $H = (V, E_H)$ be the graph with $E_H = \{\{v, w\} \mid v, w \in V,\ v \neq w,$ $\exists i \in I :\ v, w \in X_i\}$. By the second property of tree decompositions, G is a subgraph of H. By construction, each set X_i is a clique in H. So, by Theorem 4, H is triangulated. From Lemma 2, we see that the maximum size of a clique in H is at most $k+1$.

(ii) $\rightarrow$ (iii): This follows when we set π to be the perfect elimination ordering of H.

(iii) $\rightarrow$ (i): Let H be the fill-in graph of G with respect to π. By Theorem 4, there is a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of H such that each set X_i is a clique in H. So, by assumption, the width of this tree decomposition is at most k; as G is a subgraph of H, this is also a tree decomposition of G.

(iii) $\rightarrow$ (iv): Observe that the set consisting of a vertex v and the higher numbered neighbours of v in the fill-in graph form a clique in the fill-in graph.

(iv) $\rightarrow$ (iii): Suppose no vertex $v \in V$ has more than k neighbours with a higher number in π in the fill-in graph H of G with respect to π. Let W be a clique in H, and let x be the vertex in W with the smallest number in π. As all vertices in $W - \{x\}$ are higher numbered neighbours of x in H, $|W - \{x\}| \leq k$. □

Besides algorithms that construct tree decompositions directly, several construct orderings π as in Theorem 6(iv). Such algorithms will be discussed in Section 3.

3 Using elimination orderings

In this section, we look to heuristics for treewidth that are based upon building an elimination ordering. These are based on the equivalence given in Theorem 6, in particular, the characterisation of treewidth by the maximum number of higher numbered neighbours in a fill-in graph of an elimination ordering.

3.1 Tree decomposition construction

We see that Theorem 6 implies that each permutation of the vertices of a graph gives us a heuristic upper bound for the treewidth of the graph: given such elimination ordering π, we can build the fill-in graph with respect to π, and from that the corresponding tree decomposition. Thus, each algorithm that builds permutations of the vertices of a graph can be seen as an upper bound heuristic for treewidth. In this section, we discuss a number of such algorithms that have been used for this purpose, but first we briefly show how we can construct the corresponding tree decomposition directly, given an elimination ordering.

Definition 7 *Let $G = (V, E)$ be a graph, and $v \in V$ be a vertex.* Eliminating *v is the operation, that adds an edge between each pair of non-adjacent neighbours of v, and then removes v.*

In Algorithm 2, we give a recursive procedure that builds a tree decomposition from a permutation. It is not hard to turn this into an efficient iterative procedure. The following result shows correctness of the algorithm.

Lemma 8 *Let $G = (V, E)$ be a graph, and π be an elimination ordering of G. Let $H = (V, E_H)$ be the fill-in graph of G with respect to G. Suppose $V = \{v_1, \ldots, v_n\}$, and for all $v_i \in V$, $\pi(v_i) = i$. Algorithm 2 outputs, when given G and vertex ordering $(v_1, v_2, \ldots, v_n)$, a tree decomposition $(\{X_v \mid v \in V\}, T = (V, F))$, such that* *(i). For all $v_i \in V$, X_{v_i} is the set of v_i and all higher numbered neighbours of v_i in H, i.e., $X_{v_i} = \{v_i\} \cup \{v_j \mid j > i \wedge \{v_i, v_j\} \in E_H\}$.*

(ii). The width of the tree decomposition is one smaller than the maximum clique size of H.

Proof: First we note that the fill-in graph H of G with respect to G can be constructed as follows: take the graph G', obtained by eliminating v_1; let π' be the elimination ordering of G', obtained from π by removing v_1; recursively, build the fill-in graph H' of G' with respect to π', and then add v_1 and its incident edges to H'.

The result can be obtained by induction to n. The case $n = 1$ is trivial. Otherwise, as $N_G(v)$ is a clique in G', Lemma 2 guarantees that there is an $w \in V'$ with $N_G(v_1) \subseteq X_w$. We can take $w = v_j$, j the lowest numbered neighbour of v_1. As $N_G(v_1)$ is a clique in G', all vertices in $N_G(v_1) - \{v_j\}$ are neighbours of v_j in G', and hence, by induction, we have that $N_G(v_1) \subseteq X_{v_j}$.

We indeed have a tree decomposition of H and of G. Induction shows that for all $\{v_\alpha, v_\beta\} \in E_H$ with $\alpha > 1$, $\beta > 1$, there is a bag containing both v_α and v_β. By construction, v_1 and each vertex v_α with $\{v_1, v_\alpha\} \in E_H$ belongs to bag X_{v_1}. Induction shows that for all $w \in V - N_G[v]$, $I_w = \{i \in V \mid w \in X_i\}$ forms a connected subtree of the tree T. For $w \in N_G(v)$, I_w consists of a subtree of T' that contains v_j and of the new bag v_1, which is adjacent in T to v_j; thus this again forms a connected subtree of T. Finally, v_1 belongs only to bag X_{v_1}, and hence I_{v_1} forms a connected subtree of T of only one vertex.

It is not hard to see that the two stated conditions hold. For instance, each clique W is a subset of the bag of the lowest numbered vertex in W. So, we have shown the result with induction. □

Algorithm 2 PermutationToTreeDecomposition(Graph $G = (V, E)$, VertexList $(v_1, v_2, \ldots, v_n)$)

```
if $n = 1$ then
    Return a tree decomposition with one bag $X_{v_1} = \{v_1\}$.
end if
Compute the graph $G' = (V', E')$ obtained from $G$ by eliminating $v_1$.
Call PermutationToTreeDecomposition($G'$, $(v_2, v_3, \ldots, v_n)$) recursively, and let
$(\{X_w \mid w \in V'\}, T' = (V', F'))$ be the returned tree decomposition.
Let $v_j$ be the lowest numbered neighbour of $v_1$, i.e., $j = \min\{i \mid \{v_1, v_i\} \in E\}$.
Construct a bag $X_{v_1} = N_G[v_1]$.
return $(\{X_v \mid v \in V\}, T = (V, F))$ with $F = F' \cup \{v_1, v_j\}$
```

The discussion above gives us a simple general format of several treewidth heuristics. First, use some algorithm to build an ordering of the vertices of the graph, and then convert it to a tree decomposition of it by Algorithm 2. The width is the maximum number of higher numbered neighbours of a vertex in the corresponding fill-in graph.

3.2 Triangulation recognition heuristics

There are several vertex ordering algorithms that have been used for this purpose. Some of these are based upon algorithms that give a perfect elimination ordering when the input graph is a triangulated graph, and were originally proposed as algorithms to recognise triangulated graphs. One of these is the Maximum Cardinality Search algorithm by Tarjan and Yannakakis [59]. In the Maximum Cardinality Search algorithm, a vertex ordering is build from right to left. An arbitrary vertex is chosen as v_n, and then, at each step, the next vertex chosen must be one which is adjacent to an as large as possible number of already chosen vertices. In our experiments, we call this algorithm MCS-P. A variant of this algorithm, MCS-M, has been proposed by Berry et al. [9, 8]. The MCS-M algorithm guarantees that the fill-in graph H respective to the ordering obtained by the MCS-M algorithm is a *minimal* triangulation. Another algorithm to recognise triangulated graphs is the Lexicographic Breadth First Search algorithm by Rose et al. [56]. This algorithm also comes in two flavours: LEX-P and LEX-M; LEX-P is faster, whereas LEX-M guarantees again that the corresponding triangulation is minimal. Further generalizations of these algorithms have been considered by Berry et al. [13]. See also [62].

For the recognition of triangulated graphs, the result is independent of first chosen vertex v_n. For computing good tree decompositions the result heavily depends on this vertex and so it is straightforward to run these algorithms with all potential start vertices once at the cost of increasing the complexity by a factor $O(n)$.

3.3 Greedy triangulation algorithms

Other heuristics for treewidth build the ordering together with adding fill-in edges. Algorithm 3 shows the general scheme to build elimination orderings greedily. For each different criterion X, we have a different treewidth heuristic. We only give the code to build the ordering; the corresponding fill-in graphs and tree decompositions can be made as in Algorithms 1 and 2.

Algorithm 3 GreedyX(Graph $G = (V, E)$)

```
H = G;
for $i = 1$ to $n$ do
    Choose a vertex $v$ from $H$ according to criterion $X$.
    Let $v$ be the $i$th vertex in ordering $\pi$.
    Let $H$ be the graph, obtained by eliminating $v$ from $H$ (make the neighbourhood of $v$
    a clique and then remove $v$.)
end for
return ordering $\pi$
```

Using different criteria X gives us different algorithms to build elimination orderings, and hence different heuristics for treewidth. A very simple criterion, and one that performs often quite well in practice is to select a vertex of smallest degree in H, the GreedyDegree heuristic. Slightly slower, but with on average slightly better bounds in practice is the GreedyFillIn heuristic (see Section 6 and [45]). In this case, we choose a vertex that causes the smallest number of fill-in edges, i.e., a vertex that has the smallest number of pairs of non-adjacent neighbours. GreedyDegree is motivated by the fact that we create a bag of size the degree of the chosen vertex plus one, GreedyFillIn by a wish not to create many new edges, as these may cause other vertices to have high degree when eliminated.

GreedyDegree and GreedyFillIn are very simple heuristics, that appear to perform very well for many instances obtained from existing applications. For our computational evaluation in Section 6, we propose in Table 1 a few alternative greedy approaches that we have considered (here $\phi_H(v)$ denotes the fill-in by elimination v in H whereas $\delta_H(v)$ denotes the degree of v in H).

algorithm	selection of next vertex
GreedyDegree	$v = \arg\min_u \delta_H(u)$
GreedyFillIn	$v = \arg\min_u \phi_H(u)$
GreedyDegree+FillIn	$v = \arg\min_u \delta_H(u) + \phi_H(u)$
GreedySparsestSubgraph	$v = \arg\min_u \phi_H(u) - \delta_H(u)$
GreedyFillInDegree	$v = \arg\min_u \delta_H(u) + \frac{1}{n^2}\phi_H(u)$
GreedyDegreeFillIn	$v = \arg\min_u \phi_H(u) + \frac{1}{n}\delta_H(u)$

Table 1: Greedy algorithms for constructing an elimination ordering

Recently, new criteria have been proposed and investigated for the selection of vertices. The new treewidth heuristics thus obtained give in some cases improvements upon the existing heuristics.

One such criterion was proposed by Clautiaux et al. [26, 27]. Here, we compute for each vertex v first a lower bound on the treewidth of the graph obtained from H by eliminating v. The vertex is chosen which has the smallest value for the sum of twice this lower bound plus the degree in H.

Inspired by results on preprocessing graphs for treewidth computations, Bachoore and Bodlaender [4] investigated several other selection criteria. To describe these, we need a few new notions.

Definition 9 *A vertex $v \in V$ is* simplicial *in graph $G = (V, E)$, if its set of neighbours $N_G(v)$ is a clique in G. A vertex $v \in V$ is* almost simplicial*, if it has a neighbour w, such that the set of neighbours except w, $N_G(v) - \{w\}$ is a clique in G.*

If v is simplicial in $G = (V, E)$, then there exists an elimination ordering of G that starts with v and gives the optimal treewidth, i.e., for which the maximum clique size of its fill-in graph equals the treewidth of G. Thus, if v is simplicial, it seems a good choice to select v as the first vertex of the elimination ordering. Such an elimination ordering also exists, when v is almost simplicial *and* the degree of v is at most the treewidth of G [21].

Now, if we have a lower bound *low* on the treewidth of G, then we can start the ordering with an almost simplicial vertex v whose degree is at most *low*.

The Enhanced MinimumFillIn algorithm of [4] uses a lower bound *low* on the treewidth of the input graph G, and works with the following selection criterion: if there is a simplicial vertex, or an almost simplicial vertex of degree at most *low*, then that vertex is chosen, otherwise a vertex with smallest fill-in (as in the GreedyFillIn heuristic) is chosen. The other heuristics from [4] also start with selecting simplicial or low-degree almost simplicial vertices, but then use more complicated selection criteria, based upon the degree, fill-in, the minimum number of edges that must be added to make the vertex almost simplicial, and/or the ratio of some of these parameters.

3.4 Local search and genetic algorithms

Local search methods, like simulated annealing or tabu search, appear to give for many optimisation problems heuristics that give solutions close to optimal, but that use much time. A few studies have been carried out to use local search methods for solving treewidth or a related problem.

In particular, we discuss here the tabu search algorithm for treewidth by Clautiaux et al. [27]. We do not give full details here. The main idea of tabu search is the following: we start with an initial solution to the problem, and then step from this solution to a 'neighbouring' solution, i.e., one that is obtained from the first solution by performing a small change. Usually, one steps to the neighbouring solution with smallest cost. This process is repeated for some time, and the best solution found is reported. In order to avoid cycling among a small set of solutions, tabu search keeps a list of the last α encountered solutions during the search, and we forbid the algorithm to step to a solution that is already on the list.

In [27], Clautiaux et al. show how the tabu search paradigm can be successfully applied for approximating treewidth. The set of solutions is the set of elimination orderings of G, similar as for the heuristics discussed earlier in this section. While we want to optimise the corresponding width (the maximum number of higher numbered neighbours of a vertex in the fill-in graph of the elimination ordering), it is not wise to use this number as cost for the tabu search, as many neighbouring solutions will have the same width, and hence, the search is not well directed towards improvements. So, a more complicated cost function is used. For an elimination ordering π, its cost is

$$w_\pi \cdot n^2 + \sum_{v \in V} |N_\pi^+(v)|^2$$

where w_π is the treewidth corresponding to π, and $N_\pi^+(v)$ is the set of higher numbered neighbours of v in the fill-in graph corresponding to π. In this way, orderings that give tree decompositions with few large bags are preferred above orderings with the same width but with more large bags.

A simplified variant of the neighbourhood structure from [27] is the following: let two elimination orderings be neighbours if one can be obtained from the other by moving one vertex to a different position in the ordering. In this way, each solution has $\Theta(n^2)$ neighbours.

Clautiaux et al. use less neighbours per vertex in their neighbourhood structure. First, let us observe that certain changes of the elimination ordering do not change the corresponding triangulations (hence width).

Lemma 10 (Clautiaux et al. [27]) *Let π, π' be two elimination orderings of $G = (V, E)$, where π' is obtained from π by reversing the order of two successive vertices v, w. Let H be the fill-in graph with respect to π, and H' be the fill-in graph with respect to π'. If v and w are not adjacent in H, then $H = H'$.*

Using the lemma and induction, we see that when we take a permutation ordering π, and move a vertex v to a different position in π, such that there are no neighbours to v in π between the original and new position of v, then the fill-in graph does not change. As equal fill-in graphs correspond to basically the same tree decompositions, moves in the search where the fill-in graph does not change are highly undesirable. In [27], a vertex is moved to the position just after its first higher numbered neighbour in the fill-in graph, or to the position just before the last lower numbered neighbour in the fill-in graph. In this way, each solution / elimination ordering has at most $2n$ neighbouring solutions (each vertex can moved to at most two positions). See [27] for further details.

Kjærulff [40] has applied simulated annealing to solve a problem related to treewidth: a different cost measure obtained from the use of tree decomposition for the inference problem for probabilistic networks is used here. Larrañaha et al. [48] have used genetic algorithms for treewidth. Both Kjærulff [40] and Larrañaha et al. [48] use the elimination ordering representation, and two orderings neighbouring each other if they can be obtained with small changes like the moving of a vertex or exchange of two vertices; in [48], also a mechanism is used to make a cross-over between two elimination orderings.

3.5 Turning exact methods into heuristics

Several exact algorithms for computing the treewidth can be turned into a heuristic algorithm — one that does not necessarily give the exact answers, but uses less time. One such example is a branch and bound algorithm. Gogate and Dechter [34] give a branch and bound algorithm to compute the treewidth. An early halt of the algorithm (e.g., after some fixed amount of time has passed) gives an approximate solution. Dynamic programming algorithms also can be turned into a heuristic by dropping some elements from tables. This procedure has been suggested and evaluated in [17]. We plan to report on these and other exact approaches in [20].

4 Using Separators

In this section, we look at a number of heuristics that build a tree decomposition by finding a number of separators in the graph. There is a group of heuristics that follow the same strategy, which we term ‘splitting into components’; these are discussed in Section 4.1. Two other heuristics that use separators are discussed in Sections 4.2 and 4.3.

4.1 The Splitting Into Components Strategy

Several heuristics for treewidth use the same global strategy. We give a general description of the main scheme, but will not go into details for most of the separate heuristics.

The main idea is as follows: the graph is split with help of a separator; recursively, we find a tree decomposition for each part of the graph, and then these tree decompositions are ’glued’ together. The heuristics of this type have a general advantage above e.g., those based on elimination orderings: they come with guarantees on the treewidth obtained by the heuristic. They also have disadvantages: they are significantly more complex, significantly slower, and often give bounds that are higher than those of simpler algorithms.

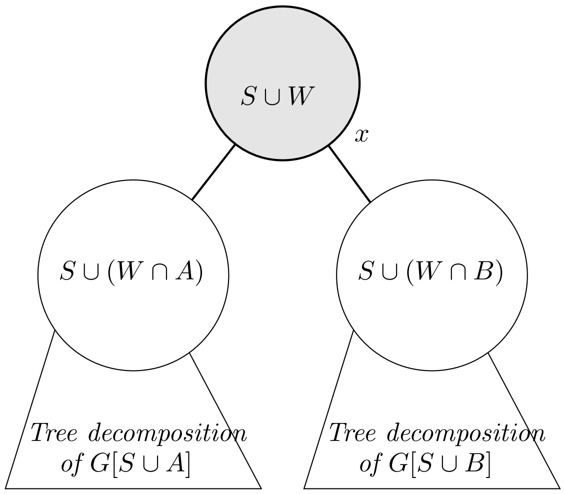
Each of [2, 1, 5, 24, 18, 29, 42, 47, 52] contains a heuristic of this type. See also [41, Chapter 10.5]. The method can be traced back to an algorithm by Robertson and Seymour in [54]. This algorithm either decides that the *branchwidth* is larger than k, or finds a branch decomposition of width at most $3k$; the algorithm uses time quadratic in n but exponential in k. Branchwidth and treewidth are closely related, and a branch decomposition can be easily converted to a tree decomposition. The algorithm given below is basically the algorithm from [54], but stated in terms of treewidth and tree decompositions instead of branchwidth and branch decompositions.

Let $S \subset V$ be a separator of $G = (V, E)$. Without loss of generality, we partition $V - S$ in vertex sets A and B such that S is a a-b-separator for all $a \in A$, $b \in B$. We say that S *separates* A from B in G. For proofs of results similar to Lemma 11, see e.g., [18, 33, 42, 50, 53].

Lemma 11 *Let $G = (V, E)$ be a graph of treewidth at most k, and let $W \subseteq V$ be a set of vertices. There is a partition of V into three sets S, A, B, such that $|S| \leq k + 1$, $|A \cap W| \leq \frac{2}{3}|W|$, $|B \cap W| \leq \frac{2}{3}|W|$, and S separates A from B.*

Suppose FindBalancedPartition is a procedure, that given a graph $G = (V, E)$ and a set $W \subseteq V$, either gives a partition of V into three sets S, A, B, fulfilling the conditions of Lemma 11, represented by the 4-tuple (true, S, A, B), or determines that such a partition does not exist, represented by (false, -,-,-). There is an implementation of FindBalancedPartition that takes $O(3^{|W|} \cdot k \cdot (n + m))$ time: we try each of the $3^{|W|}$ possibilities to distribute the vertices of W over S, A, and B; for each such possibilities we test if there is a separator of size $k - |S \cap W|$ that separates the set of vertices $A \cap W$ from the set of vertices $B \cap W$ in the graph $G[V - (S \cap W)]$; this test can be done in $O(k(n + m))$ time with flow techniques (see e.g., [31]).

Using this procedure FindBalancedPartition, Algorithm 4 describes the general scheme of this type of treewidth heuristic. It returns a tree decomposition of G with a specific bag identified as root node. Figure 2 illustrates the construction of the algorithm. The step where we added a vertex from $V - W$ to S in case ($A = \emptyset$ or $B = \emptyset$) and $S \subseteq W$ is needed to prevent the possibility of a not terminating recursive program.



$S \cup W$
x
$S \cup (W \cap A)$
$S \cup (W \cap B)$
Tree decomposition of $G[S \cup A]$
Tree decomposition of $G[S \cup B]$

Figure 2: The construction of Algorithm 4.

Theorem 12 *(i). Let $G = (V, E)$ be a graph with treewidth at most k, and suppose $|W| \leq 3k+3$. The procedure* BuildTreeDecomposition(G,W) *outputs a tree decomposition of G of width at most $4k+3$, such that a root bag of the tree decomposition contains all vertices in W.*

(ii). If procedure BuildTreeDecomposition(G,W) *outputs that the treewidth is larger than k, then the treewidth of G is at least $k+1$.*

Proof: (i) First note that the algorithm terminates: either the graph in the first argument of a recursive call has fewer vertices, or it has the same number of vertices but the number of vertices not belonging to the set of the second argument has decreased.

Now, to proof the result, we use induction to the depth of the recursion. If we return a tree decomposition with one bag, the result clearly holds.

Suppose $W \neq V$. We have that $|W \cap A| \leq \frac{2}{3}|W| \leq 2k+2$. So, $|S \cup (W \cap A)| \leq 3k+3$, and similarly, $|S \cup (W \cap B)| \leq 3k+3$. With induction, we have that the recursive calls yield tree decompositions of $G[S \cup A]$ and $G[S \cup B]$ of width at most $4k+3$, whose root bags contain respectively the vertices in $S \cup (W \cap A)$ and $S \cup (W \cap B)$. We can now verify that the algorithm indeed outputs a tree decomposition of G of width at most $4k+3$ whose root bag contains W. It is easy to observe that each vertex in V belongs to at least one bag.

Consider an edge $\{v, w\} \in E$. By the assumption on partitions, we have that $v, w \in S \cup A$ or $v, w \in S \cup B$. In the former case, $\{v, w\}$ is an edge in $G[S \cup A]$, and hence there is a bag in the tree decomposition of $G[S \cup A]$ containing both v and w; the latter case is similar.

Consider a vertex $v \in V$. Consider the set of bags containing v, I_v. If $v \in V - (S \cup A \cup W)$ then v only appears in bags in the tree decomposition of $G[V - B]$, and thus I_v forms a subtree. Similarly when $v \in V - (S \cup B \cup W)$. The remaining case is that

Algorithm 4 BuildTreeDecomposition(Graph $G = (V, E)$, VertexSet W)

```
if $W = V$ then
  return A tree decomposition with one bag containing all vertices
end if
(t, $S$, $A$, $B$) = FindBalancedPartition($G = (V, E)$, $W$)
if t $\equiv$ false then
  return Reject: treewidth is larger than $k$
end if
if ($A = \emptyset$ or $B = \emptyset$) and $S \subseteq W$ then
  Add a vertex from $V - W$ to $S$
end if
Run BuildTreeDecomposition($G[S \cup A]$, $S \cup (W \cap A)$)
Run BuildTreeDecomposition($G[S \cup B]$, $S \cup (W \cap B)$)
if at least one of these runs rejects then
  return Reject: treewidth is larger than $k$
end if
Take the disjoint union of the two recursively obtained tree decompositions
Add a new bag $x$ containing the vertices in $S \cup W$
Make $x$ adjacent to the root nodes of the two recursively obtained tree decompositions
return The just computed tree decomposition of $G$ with $x$ as root
```

$v \in (S \cup A \cup W) \cap (S \cup B \cup W) = S \cup W$. v either belongs to no bags in the tree decomposition of $G[S \cup A]$ or a connected set of bags that includes the root bag; similar, it belongs to either no bags in the tree decomposition of $G[S \cup B]$ or a connected set of bags that includes the root bag; and it belongs to bag x. Thus, these bags form a connected subtree.

We have now verified that the algorithm outputs a tree decomposition of G. Clearly, the root bag contains W. The maximum size of a bag is at most $4k+4$, by the assumptions on the widths of the recursively obtained tree decompositions and the fact that $|S \cup W| \leq 4k + 4$.

(ii) If the algorithm outputs that the treewidth is more than k, then the graph at hand has treewidth more than k by Lemma 11. As each recursive call works with an induced subgraph, and the treewidth cannot increase by taking induced subgraphs, the treewidth of the original input graph G is also larger than k. $\square$

Several variations and improvements on the method are possible. E.g., instead of working with separators that partition into two parts, one can also work with separators which partition into more parts, often guaranteeing that each part contains at most half of the vertices of W (and thus getting better treewidth bounds). Thus, different algorithms can vary in quality of obtained approximations and running times. For some algorithms in this vein, see e.g., [2, 1, 5].

4.2 The Minimum Separating Vertex Sets Heuristic

The Minimum Separating Vertex Sets (MSVS) heuristic of Koster [43] (see also [46]) refines a tree decomposition by replacing one bag with multiple smaller bags with the help of minimum separators, also called *minimum separating vertex sets*, hence the name of this heuristic. In its original version, it starts with a trivial tree decomposition (i.e., a tree decomposition with one bag $X_i = V$) but the general scheme can be applied to any tree decomposition, is described in Algorithm 5, and illustrated in Figure 3.

Algorithm 5 RefineTreeDecomposition(Graph $G = (V, E)$, TreeDecomposition $(\{X_i, i \in I\}, T = (I, F))$)

```
while $\exists i \in I$ such that $|X_i|$ maximal and $G[X_i]$ does not induce a clique do
    Construct graph $H_i$ with vertex set $X_i$ and edge set $\{\{v,w\} \in X_i \times X_i | \{v,w\} \in E \vee \exists j \neq i : v,w \in X_j\}$
    Compute minimum separator $S \subset X_i$ in $H_i$; let $W_1, \ldots, W_r$ define the $r$ connected components of $H_i[X_i - S]$
    Set $I' = I - \{i\} \cup \{i_0, \ldots, i_r\}$
    Set $X'_j = X_j$ for all $j \neq i$, $X'_{i_0} = S$, $X'_{i_q} = W_q \cup S$ for $q = 1, \ldots, r$
    Set $F' = F - \{\{i,j\} | j \in N_T(i)\} \cup \{\{i_0, i_q\} | q = 1, \ldots, r\} \cup \{\{j, i_{q_j}\} | j \in N_T(i)\}$ where $q_j \in \{1, \ldots, r\}$ such that $X_i \cap X_j \subseteq W_{q_j} \cup S$
end while
return Tree decomposition $(\{X'_j, j \in I'\}, T' = (I', F'))$
```

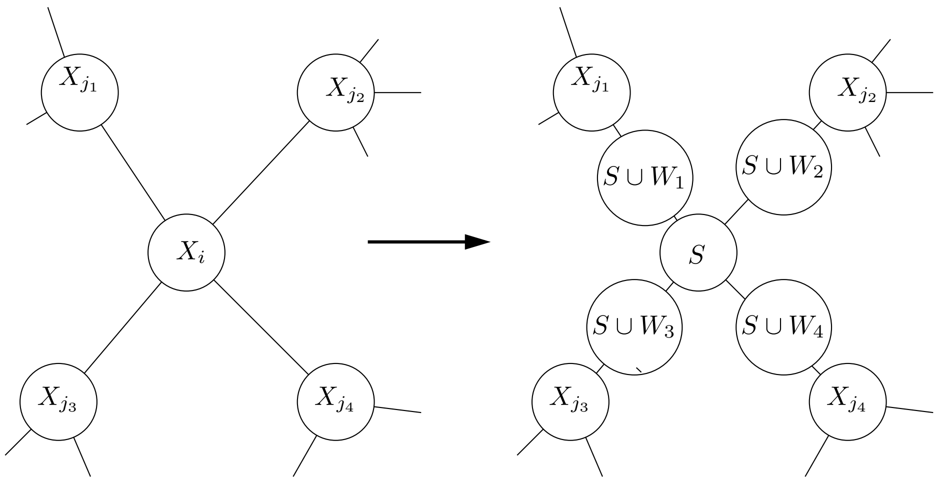


Figure 3: A refinement step in the MSVS heuristic

Unless all X_i induce complete graphs, the tree decomposition can be refined to one with smaller width; this process is repeated until the nodes of maximum cardinality in the tree decomposition cannot be refined anymore. The refinement step consists of *splitting* a bag X_i into a number of bags, each of smaller cardinality.

Suppose we have a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of $G = (V, E)$ and we want to refine bag X_i, $i \in I$. This is done by first building an auxiliary graph H_i, next finding a minimum separator S in H_i, and finally adapting the tree decomposition as dictated by S, as described below.

H_i is a graph with vertex set X_i. For each pair of vertices $v, w \in X_i$, $v \neq w$, we take an edge $\{v, w\}$ in H_i, if and only if v and w are adjacent in G or there is a node $j \neq i$ with $v, w \in X_j$.

The second step is finding a minimum separator in H_i. Finding a minimum separator in a graph with n vertices, m edges, can be done in $O(\max\{k^3 \cdot m, k \cdot n \cdot m\})$ time, with k the size of the minimum separator with help of network flow techniques, see e.g., [31, Section 6.2]. If H_i would be a clique, then we cannot refine i and therefore this case is not considered.

Now, if we have separator S in H_i, we can refine X_i as follows. The graph $H_i[X_i - S]$ has at least two connected components, say these connected components have vertex sets $W_1, \ldots, W_r$. The refinement of X_i takes place as follows. Each node $j \neq i$ is kept, setting $X'_j = X_j$. i is replaced by $r + 1$ nodes $i_0, \ldots, i_r$, with $X'_{i_0} = S$, and for $1 \leq q \leq r$, $X'_{i_q} = S \cup W_q$. In the new tree T', we keep all edges between nodes $\neq i$. Then we make i_0 adjacent to each i_q, $1 \leq q \leq r$. Each neighbour j of i is made adjacent to one of the nodes i_q, in the following way. Consider $Y_j = X_i \cap X_j$. Note that Y_j is a clique in H_i. Thus, there cannot be two different connected components of $H_i[X_i - S]$ that both contain vertices of Y_j, and hence there must be a q_j, $1 \leq q_j \leq r$, such that $Y_j \subseteq W_{q_j} \cup S$. Make node j adjacent to node i_{q_j} in T. Let $(T' = (I', F'), \{X'_j : j \in I'\})$ be the resulting structure, which is a tree decomposition by the following lemma.

Lemma 13 *Let $(T' = (I', F'), \{X'_j : j \in I'\})$ be obtained from applying a refinement step to node $i \in I$ in a tree decomposition $(T = (I, F), \{X_j : j \in I\})$ of graph $G = (V, E)$. Then $(T' = (I', F'), \{X'_j : j \in I'\})$ is a tree decomposition of G.*

Proof: It is trivial that the first of the three conditions of tree decomposition is fulfilled. Consider an edge $\{v, w\} \in E$. There is a node $j \in I$ with $v, w \in X_j$. If $j \neq i$, then $v, w \in X'_j$ also after the refinement step. Otherwise, $\{v, w\}$ is an edge in H_i. If $v, w \in S$, then $v, w \in X'_{i_0}$. If $v \in W_q$, $1 \leq q \leq r$, then $w \in S \cup W_q$, so $v, w \in X'_{i_q}$; similarly when $w \in W_q$. We have now verified the second requirement of tree decompositions.

Consider a vertex $v \in V$, and the sets of nodes $I_v = \{j \in I : v \in X_j\}$, $I'_v = \{j \in I' : v \in X'_j\}$. If $v \notin X_i$, then $I'_v = I_v$, hence I'_v is connected in T'. If $v \in S$, then $I'_v = I_v - \{i\} \cup \{i_0, \ldots, i_r\}$, and one easily sees that the connectedness of I'_v follows from the connectedness of I_v. If $v \in W_q$, $1 \leq q \leq r$, then for every neighbour j of i with $v \in X_j$, we have that j is adjacent to i_q in T'. As $I'_v = I_v - \{i\} \cup \{i_q\}$, connectedness of I'_v again follows. We can now conclude that $(\{X'_j : j \in I'\}, T' = (I', F'))$ is indeed a tree decomposition of G. □

Note that each of $X'_{i_0}, \ldots, X'_{i_r}$ is of smaller cardinality than X_i. A refinement step is illustrated in Figure 3. If H_i is a complete graph, then it has no separating vertex set. If H_i is not complete, then it has: when v and w are not adjacent in H_i, then $X_i - \{v, w\}$ is a separating vertex set of H_i. Thus, as long as there is a node $i \in I$ in the tree decomposition of maximum cardinality with H_i not a complete graph, the refinement step can be applied. When no refinements are possible, the MSVS heuristic stops.

4.3 Completing Minimal Separators

We can look at the MSVS heuristic in a different way, and arrive at a heuristic that builds a minimal triangulation of a graph by adding edges to vertices in minimal separators.

Consider again the MSVS heuristic, and the triangulation H of G obtained by making each set X_i in the final tree decomposition a clique. Consider a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ that is used in an intermediate step during the algorithm. For some pairs of vertices $v, w \in V$, we can deduce that they must form an edge in H, namely when $\{v, w\}$ is already an edge in G, or if there are at least two bags X_i, X_j, with $v, w \in X_i$ and $v, w \in X_j$. Say G' is the graph, formed by these edges, i.e., $\{v, w\}$ is an edge in G', if it is an edge in G, or there are at least two bags that contain both v and w. In one refinement step, we precisely add those edges to G' that turn S into a clique. Following terminology of [24], let *completing* a vertex set S in a graph G be the operation that turns S into a clique, i.e., for each pair of vertices v, w, $v \neq w$ in S, we add an edge from v to w to G, unless v and w were already adjacent.

Thus, in the MSVS heuristic, we repeatedly build a graph H_i, and then complement a minimum size separator in H_i. In [24], a heuristic is described that works slightly differently: we now complement a minimal or minimum size separator in G'.

Algorithm 6 MINIMALTRIANGULATION(GRAPH $G = (V, E)$)

```
$G' = G$;
**while** $G'$ is not a triangulated graph **do**
  Choose a minimal separator $S$ in $G'$ that is not a clique.
  Let $G'$ be the graph, obtained by completing $S$ in $G'$.
**end while**
**return** $G'$
```

In Algorithm 6 we see the main scheme of Bouchitté et al. [24]. In a refinement, we always choose a set S of minimum size that is not a clique. In [24], it is discussed how the algorithm can be carried out, and implemented to run in $O(n^{5.5})$ time. In [24], it is shown that this algorithm approximates the treewidth within a multiplicative factor of $8a$, where a is the asteroidal number of G. No experimental evaluation of this algorithm is known to us.

The similarity between the MinimalTriangulation heuristic and the MSVS heuristic can be stressed further by observing that each separator in the graph H_i as built by the MSVS heuristic is also a separator in G'.

5 Postprocessing

For several treewidth heuristics, it is the case that the triangulation that corresponds to the tree decomposition (compare Theorem 6) is not always a *minimal* triangulation. In that case, it may be useful to apply a postprocessing step, using a subroutine that solves the *triangulation minimisation* problem.

In the triangulation minimisation problem, we are given a graph $G = (V, E)$ and a triangulation $H = (V, F)$ of G, and we look for a minimal triangulation $G' = (V, E')$ of G, with the property that G' is a subgraph of H, i.e., G' is a triangulated graph and $E \subseteq E' \subseteq F$. There are several algorithms known that solve the triangulation minimisation problem in $O(nm)$ time. In our experiments, we used the algorithm of Blair et al. [14]. Other algorithms that solve this problem can be found in [7, 10, 28, 39]. See also [11, 12, 51], and the overview paper by Heggernes [37]. Recently, Heggernes et al. [38] found a faster algorithm for the triangulation minimisation problem.

Using triangulation minimisation, we can do the following: we run some heuristic that produces tree decompositions. The tree decomposition is converted to a triangulation H of the input graph G. Given G and H, we find a minimal triangulation G' of G that is a subgraph of H using an algorithm for the triangulation minimisation problem. This minimal triangulation is converted back to a tree decomposition of G.

This postprocessing step can never increase the treewidth, but will for some instances decrease it.

6 Computational Evaluation

In this section, we present a computational evaluation of a selection of the algorithms presented. We in particular focus on the variants of GreedyX as those turn out to provide the best value for money (i.e., time). First, we discuss the experimental setup, and the next the results.

6.1 Experimental setup

All algorithms that have been evaluated have been implemented in C++ with use of the Boost Graph Library [58].

For most of the presented algorithms, computational studies have been presented in the corresponding publications. Typically the algorithms have been tested on a number of graphs originating from a variety of applications. In addition, sometimes graphs with a well-known combinatorial structure like the Petersen graph have been considered. Although we strongly agree that the performance of the upper bound algorithms on graphs originating from applications is of utmost importance, we follow another approach in the paper. For most graphs from applications it is difficult to determine the optimal width, and so the quality of the algorithms is masked by this fact.

To evaluate a selection of the upper bound algorithms on their scientific merits, we adapted the procedure proposed by Shoikhet and Geiger [57] and test the algorithms on randomly generated partial-k-trees. A k-tree is a triangulated graph with the property that there exists a perfect elimination ordering π with $|N_{\pi}^{+}(v)| = \min\{k, n - \pi(v)\}$ for all $v \in V$ (note that $n - \pi(v)$ is the degree of the last k vertices in the ordering). Thus, a k-tree has exactly $k(k-1)/2 + k(n-k) = kn - k(k+1)/2$ edges. A partial-k-tree G is a graph for which there exist a k-tree supergraph. Hence, the treewidth of a partial-k-tree is at most k. If we in addition can guarantee that the treewidth of a randomly generated partial-k-tree is at least k, the treewidth equals k and we have a good basis to compare upper bound algorithms.

Randomly generated partial-k-trees are characterised by three parameters: the number of vertices n, the value k, and the number of edges missing to be a k-tree as percentage p of the number of k-tree edges. All combinations of $n \in \{100, 200, 500, 1000\}$, $k \in \{10, 20\}$ and $p \in \{30, 40, 50\}$ have been considered. For every choice (n, k, p) of the parameters, we generate 50 instances by first constructing a k-tree and then removing randomly $p\%$ of the edges (hence, the total number of graphs is 1200).

To assure that the treewidth of a randomly generated partial-k-tree is at least k, we apply the maximum minimum degree heuristic for the contraction degeneracy lower bound [22]. For the above parameter choices, this lower bound matches k without exception.

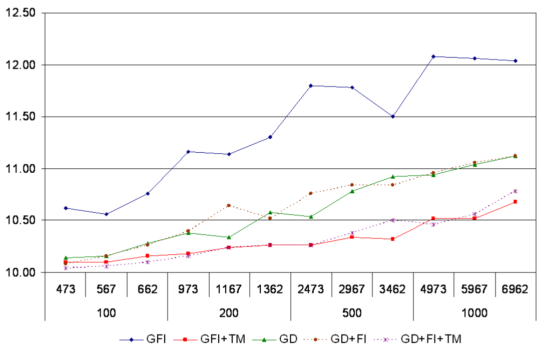
6.2 Comparison of greedy algorithms

In Table 1 on page 9, a number of algorithms to construct an elimination ordering greedily have been presented. Experiments revealed that the GreedySparsestSubgraph is not competitive. The idea to find the sparsest subgraph might be attractive on first sight but implies that we have a preference to select vertices of high degree as long as the fill-in is relatively low for those. The degree however will determine the width of the tree decomposition in the end and thus high degree vertices are not a good choice in this respect. Therefore, we leave this algorithm out of our further discussion.

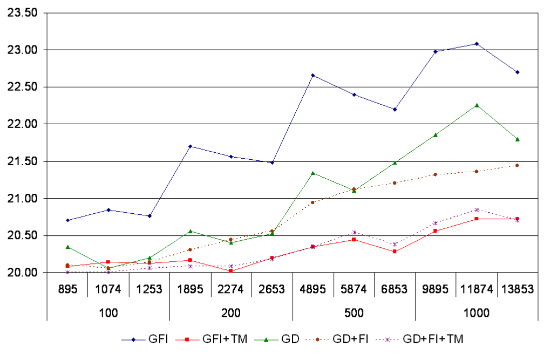
For the variants GreedyFillInDegree and GreedyDegreeFillIn (of respectively GreedyDegree and GreedyFillIn) where a second criterion is used as tie breaker, the differences with the algorithms without tie breaker turn out to marginal (slightly better in most, but not all cases). Therefore we focus on the algorithms without tie breaker in the sequel.

Figure 4 shows the average width obtained with the algorithms GreedyFillIn, Greedy-Degree, and GreedyDegree+FillIn for the different parameter settings. In addition, the results of the postprocessing step for two of the algorithms are shown. The postprocessing step did not have any effect on the results of the GreedyDegree algorithm. In fact, the width could not be improved for a single instance (the number of fill-in edges could be reduced in a few cases and hence the orderings are not providing minimal triangulations in general). The distribution of the width is shown in Figure 5(a).

The figures show that GreedyFillIn is outperformed by GreedyDegree, but that

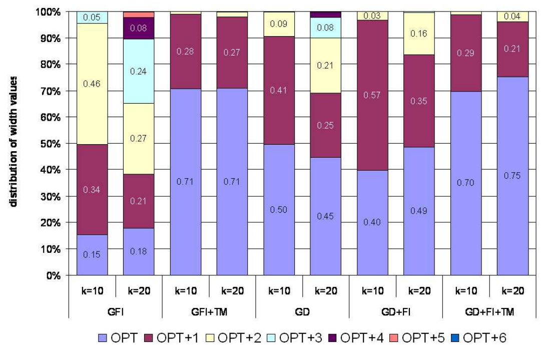


(a) $k = 10$

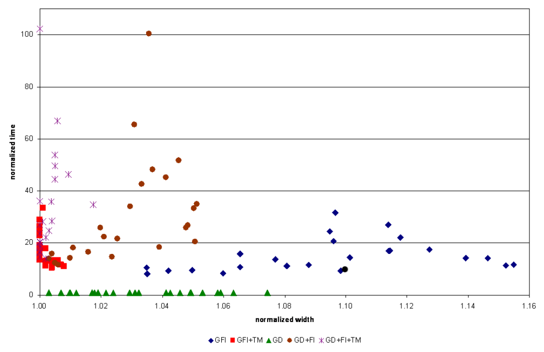


(b) $k = 20$

Figure 4: Performance of greedy algorithms: average width obtained for samples of 50 randomly generated partial k graphs (treewidth equals k in all cases). The size of the graphs in number of vertices and edges is given on the x-axis; the average width on the y-axis. GFI = GreedyFillIn, GD = GreedyDegree, GD+FI = GreedyDegree+FillIn, +TM = Triangulation minimisation algorithm applied on result of greedy algorithm.



(a) distribution of width values



(b) normalised width vs. normalised time

Figure 5: Performance of greedy algorithms. Figure (a) shows the distribution of the width relative to the optimal value (OPT). Figure (b) shows the performance of the algorithms in relation to the computation time. Both times and widths are normalised according to minimum achieved value (not the optimal width). See Figure 4 for explanation of algorithm acronyms.

after triangulation minimization, the better results are achieved by GreedyFillIn. The combination of GreedyFillIn and GreedyDegree in GreedyDegree+FillInturns out to be competitive and for $k = 20$ slightly better than GreedyDegree. Adding triangulation minimization to this algorithm results in results that are similar to GreedyFillIn with triangulation minimization.

Besides the quality of the results, also the computation times play an important role by selecting the best algorithm. Figure 5(b) shows a scatter plot of the normalised average width values and the normalised average computation times. The normalization is done individually for every triple (n, k, p) with respect to the smallest computation time and the best average width. The plot shows that GreedyDegree is the clear winner in computation time.

6.3 Triangulation recognition heuristics

One might think that the triangulation recognition heuristics like Maximum Cardinality Search (MCS/MCS-M) and Lexicographic Breadth First Search (LEX-P/LEX-M) should perform better than the greedy algorithms as they incorporate more graph theoretical knowledge. The truth is quite contrary. Both for graphs of practical applications and the randomly generated partial-k-trees, the results for the adapted recognition heuristics are not even close to those of the greedy algorithms, regardless the starting vertex selected. To illustrate this, we performed two experiments.

First, we took one of the randomly generated partial-k-trees with $n = 100$ vertices, $k = 10$ and $p = 30$. For this graph, we ran the algorithms MCS, MCS-M, LEX-P, and LEX-M for all possible start vertices. Figure 6 shows the resulting widths for the four algorithms, displayed against the degree of the start vertex. It is clear from Figure 6

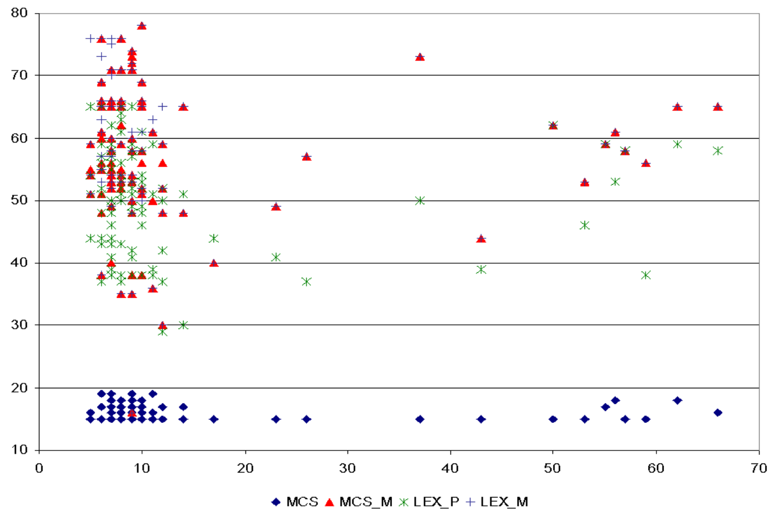


Figure 6: Resulting width of the triangulation recognition heuristics compared to degree of the start vertex

that the best choice is the MCS algorithm. But even with this algorithm the best width obtained is 15, whereas the GFI found the optimal width of 10 for this instance. Although MCS-M and LEX-M found an elimination ordering with width 16, the overall performance of the algorithms was significantly worse than MCS (the best width by LEX-P was 29). Remarkable, the highest reported width is achieved by minimal triangulation variants. For both MCS-M and LEX-M there exists a start vertex resulting in a minimal triangulation of width 78.

The above results are not due to an unfortunate choice of the partial-k-tree as a second experiment showed. For all randomly generated partial-k-trees with $n = 100$ and $k = 10$ used in the previous subsection, we ran MCS. The results are reported in Table 2 and clearly show that MCS is outperformed by GFI in all cases.

$n = 100$, $k = 10$	GFI	GFI+TM	MCS	MCS+TM	MCS-M
$p = 50$	10.62	10.10	16.64	11.84	27.52
$p = 40$	10.56	10.10	16.08	11.78	26.38
$p = 30$	10.76	10.16	14.72	11.32	22.30

Table 2: Average width by GFI, MCS, and MCS-M, with and without triangulation minimization

Since MCS does not provide a minimal triangulation, the procedure of Section 5 can be applied on the best elimination ordering generated. The average width after triangulation minimisation is also reported in Table 2 together with the result for MCS-M which include the triangulation minimization. The results show that (i) triangulation minimization significantly reduces the width of the MCS-generated elimination orderings, (ii) MCS-M is outperformed by this procedure, (iii) the width is still not competitive with the width obtained via GFI, with or without triangulation minimization.

To understand better why the triangulation recognition heuristics perform so badly for relatively small graphs, we have set up a final experiment. We first randomly generate a k-tree with $n = 100$ and $k = 10$. Next, we randomly remove edges in steps of 1% until 10% of the edges have been removed. For each of the 11 graphs generated this way, we count how many start vertices result in a certain width. The results are show in Table 3.

Since the MCS algorithm can start with any vertex for triangulated graphs, it is not a surprise that the optimal width is found in all cases if no edges are removed. However, as soon as 2% of the edges are removed, none of the start vertices results in the optimal width. GFI, in contrast, still finds the optimal width after removal of 10% edges.

7 Conclusions

In this paper, we discussed several upper bound heuristics for treewidth. Each of the heuristics finds for a given graph $G = (V, E)$ a tree decomposition of G.

Experiments show that in many cases, the heuristics perform reasonably well, i.e., give tree decompositions whose width comes close to (and sometimes equals) the exact

edges removed (%)	0	1	2	3	4	5	6	7	8	9	10
width											
10	100	100									
11			70								
12			30	97	28						
13				3	44	53	91	74	79	69	42
14					28	47	9	26	21	24	8
15										7	46
16											4

Table 3: Histograms of returned widths by MCS after removal of a percentage of the edges of a 10-tree using all possible 100 start vertices

treewidth. Some of the well performing heuristics are very fast. Thus, one can conclude that for many practical purposes, there are good methods to find tree decompositions with small treewidth.

Which of the heuristics is actually the best depends on the application. Our experiments show that different heuristics have different sets of instances on which they perform particularly well, see [61].

In many cases, the algorithms that run on the tree decompositions have a running time that is exponential in the width. Depending on the precise running times of this algorithm, and the number of times the same tree decomposition is used for different computations, it can make sense to spend more time on finding a good tree decomposition, e.g., to use a slower algorithm that computes the treewidth exactly. An overview of such algorithms is planned [20].

Many theoretical studies on treewidth give algorithms that start with the linear time algorithm for finding tree decompositions of width at most k for fixed k from [15]. While this often gives the theoretically best asymptotic bound, this is not what one would do in practice: in a real life setting, one would instead use one of the heuristics discussed in this paper, or run a few of the heuristics and take the best solution found. For this purpose, we are developing an interactive website [44], that allows to experiment with a number of the discussed algorithms on graphs of your choice. The algorithms take as input a graph in the standardised DIMACS format [30] and outputs the width found as well as the corresponding elimination ordering.

References

[1] E. Amir. Approximation algorithms for treewidth. Algorithmica, published online 2008.

[2] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 7–15,

2001.

[3] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Disc. Appl. Math.*, 23:11–24, 1989.

[4] E. H. Bachoore and H. L. Bodlaender. New upper bound heuristics for treewidth. In S. E. Nikoletseas, editor, *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms WEA 2005*, pages 217–227. Springer Verlag, Lecture Notes in Computer Science, vol. 3503, 2005.

[5] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125:3–17, 2001.

[6] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8:216–235, 1987.

[7] A. Berry. A wide-range efficient algorithm for minimal triangulation. In *SODA'99: ACM-SIAM Symposium on Discrete Algorithms*, pages 860–861, 1999.

[8] A. Berry, J. Blair, P. Heggernes, and B. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39:287–298, 2004.

[9] A. Berry, J. R. S. Blair, and P. Heggernes. Maximum cardinality search for computing minimal triangulations. In L. Kučera, editor, *Proceedings 28th Int. Workshop on Graph Theoretic Concepts in Computer Science, WG'02*, pages 1–12. Springer Verlag, Lecture Notes in Computer Science, vol. 2573, 2002.

[10] A. Berry, J.-P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *J. Algorithms*, 58:33–66, 2006.

[11] A. Berry, P. Heggernes, and G. Simonet. The minimum degree heuristic and the minimal triangulation process. In *Proceedings 29th Int. Workshop on Graph Theoretic Concepts in Computer Science, WG'03*, pages 58–70. Springer Verlag, Lecture Notes in Computer Science, vol. 2880, 2003.

[12] A. Berry, P. Heggernes, and Y. Villanger. A vertex incremental approach for maintaining chordality. *Disc. Math.*, 306:318–336, 2006.

[13] A. Berry, R. Krueger, and G. Simonet. Ultimate generalizations of LexBFS and LEX M. In *Proceedings 31st International Workshop on Graph-Theoretic Concepts in Computer Science WG'05*, pages 199–213. Springer Verlag, Lecture Notes in Computer Science, vol. 3787, 2005.

[14] J. R. S. Blair, P. Heggernes, and J. Telle. A practical algorithm for making filled graphs minimal. *Theor. Comp. Sc.*, 250:125–141, 2001.

[15] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[16] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc.*, 209:1–45, 1998.

[17] H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos. On exact algorithms for treewidth. In Y. Azar and T. Erlebach, editors, *Proceedings 14th Annual European Symposium on Algorithms, ESA 2006*, pages 672–683. Springer Verlag, Lecture Notes in Computer Science, vol. 4168, 2006.

[18] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and minimum elimination tree height. *J. Algorithms*, 18:238–255, 1995.

[19] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations II. Lower bounds. Paper in preparation, 2008.

[20] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations III. Exact algorithms and preprocessing. Paper in preparation, 2008.

[21] H. L. Bodlaender, A. M. C. A. Koster, and F. v. d. Eijkhof. Pre-processing rules for triangulation of probabilistic networks. *Computational Intelligence*, 21(3):286–305, 2005.

[22] H. L. Bodlaender, A. M. C. A. Koster, and T. Wolle. Contraction and treewidth lower bounds. *J. Graph Algorithms and Applications*, 10:5–49, 2006.

[23] H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Disc. Math.*, 6:181–188, 1993.

[24] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Disc. Appl. Math.*, 136:183–196, 2004.

[25] H. Chen. Quantified constraint satisfaction and bounded treewidth. In R. L. de Mántaras and L. Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004*, pages 161–165, 2004.

[26] F. Clautiaux, J. Carlier, A. Moukrim, and S. Négre. New lower and upper bounds for graph treewidth. In J. D. P. Rolim, editor, *Proceedings International Workshop on Experimental and Efficient Algorithms, WEA 2003*, pages 70–80. Springer Verlag, Lecture Notes in Computer Science, vol. 2647, 2003.

[27] F. Clautiaux, A. Moukrim, S. Négre, and J. Carlier. Heuristic and meta-heuristic methods for computing graph treewidth. *RAIRO Operations Research*, 38:13–26, 2004.

[28] E. Dahlhaus. Minimal elimination ordering inside a given chordal graph. In *Proceedings 23rd International Workshop on Graph-Theoretic Concepts in Computer Science WG'97*, pages 132–143. Springer Verlag, Lecture Notes in Computer Science, vol. 1335, 1997.

[29] R. Diestel, T. R. Jensen, K. Y. Gorbunov, and C. Thomassen. Highly connected sets and the excluded grid theorem. *J. Comb. Theory Series B*, 75:61–73, 1999.

[30] The second DIMACS implementation challenge: NP-Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability. See http://dimacs.rutgers.edu/Challenges/, 1992–1993.

[31] S. Even. *Graph Algorithms.* Pitman, London, 1979.

[32] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Comb. Theory Series B*, 16:47–56, 1974.

[33] J. R. Gilbert, D. J. Rose, and A. Edenbrandt. A separator theorem for chordal graphs. *SIAM J. Alg. Disc. Meth.*, 5:306–313, 1984.

[34] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence UAI-04*, pages 201–208, Arlington, Virginia, USA, 2004. AUAI Press.

[35] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[36] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural csp decomposition methods. *Acta Informatica*, 124:243–282, 2000.

[37] P. Heggernes. Minimal triangulations of graphs: A survey. *Disc. Math.*, 306:297–317, 2006.

[38] P. Heggernes, J. A. Telle, and Y. Villanger. Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. *SIAM J. Disc. Math.*, 19:900–913, 2005.

[39] P. Heggernes and Y. Villanger. Efficient implementation of a minimal triangulation algorithm. In R. Möhring and R. Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms, ESA'2002*, pages 550–561. Springer Verlag, Lecture Notes in Computer Science, vol. 2461, 2002.

[40] U. Kjærulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 2:2–17, 1992.

[41] J. Kleinberg and Éva Tardos. *Algorithm Design.* Addison-Wesley, Boston, 2005.

[42] T. Kloks. *Treewidth. Computations and Approximations.* Lecture Notes in Computer Science, Vol. 842. Springer-Verlag, Berlin, 1994.

[43] A. M. C. A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, Univ. Maastricht, Maastricht, The Netherlands, 1999.

[44] A. M. C. A. Koster. Treewidth optimization interface. In preparation, 2008.

[45] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. van Hoesel. Treewidth: Computational experiments. In H. Broersma, U. Faigle, J. Hurink, and S. Pickl, editors, *Electronic Notes in Discrete Mathematics*, volume 8, pages 54–57. Elsevier Science Publishers, 2001.

[46] A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40(3):170–180, 2002.

[47] J. Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, 20:20–44, 1996.

[48] P. Larrañaga, C. M. H. Kuijpers, M. Poza, and R. H. Murga. Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing (UK)*, 7(1):19–34, 1997.

[49] S. J. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *The Journal of the Royal Statistical Society. Series B (Methodological)*, 50:157–224, 1988.

[50] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Analysis and Applications*, 11:134–172, 1990.

[51] B. W. Peyton. Minimal orderings revisited. *SIAM J. Matrix Anal. Appl.*, 23:271–294, 2001.

[52] B. Reed. Finding approximate separators and computing tree-width quickly. In *Proceedings of the 24th Annual Symposium on Theory of Computing*, pages 221–228, New York, 1992. ACM Press.

[53] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.

[54] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory Series B*, 63:65–110, 1995.

[55] H. Röhrig. Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1998.

[56] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.

[57] K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. In *Proc. National Conference on Artificial Intelligence (AAAI '97)*, pages 185–190. Morgan Kaufmann, 1997.

[58] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual.* Addison-Wesley Professional, 2001. Software available on `http://www.boost.org`.

[59] R. E. Tarjan and M. Yannakakis. Simple linear time algorithms to test chordiality of graphs, test acyclicity of graphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13:566–579, 1984.

[60] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Disc. Math.*, 10:529 – 550, 1997.

[61] Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib, 2004– . . . .

[62] Y. Villanger. Lex M versus MCS-M. *Disc. Math.*, 306:393–400, 2006.

[63] T. V. Wimer, S. T. Hedetniemi, and R. Laskar. A methodology for constructing linear graph algorithms. *Congressus Numerantium*, 50:43–60, 1985.

[64] J. Zhao, D. Che, and L. Cai. Comparative pathway annotation with protein-DNA interaction and operon information via graph tree decomposition. In *Proceedings of Pacific Symposium on Biocomputing (PSB 2007)*, volume 12, pages 496–507, 2007.

[65] J. Zhao, R. L. Malmberg, and L. Cai. Rapid ab initio prediction of RNA pseudoknots via graph tree decomposition. *Journal of Mathematical Biology*, 56(1–2):145–159, 2008.