Treewidth: Computational Experiments

Arie M.C.A. Koster[†] Hans L. Bodlaender[‡] Stan P.M. van Hoesel[§]

December 28, 2001

Abstract

Many $\mathcal{NP}$-hard graph problems can be solved in polynomial time for graphs with bounded treewidth. Equivalent results are known for pathwidth and branchwidth. In recent years, several studies have shown that this result is not only of theoretical interest but can successfully be applied to find (almost) optimal solutions or lower bounds for many optimization problems.

To apply a tree decomposition approach, the treewidth of the graph has to be determined, independently of the application at hand. Although for fixed k, linear time algorithms exist to solve the decision problem "treewidth $\leq k$", their practical use is very limited. The computational tractability of treewidth has been rarely studied so far. In this paper, we compare four heuristics and two lower bounds for instances from applications such as the frequency assignment problem and the vertex coloring problem.

Three of the heuristics are based on well-known algorithms to recognize triangulated graphs. The fourth heuristic recursively improves a tree decomposition by the computation of minimal separating vertex sets in subgraphs. Lower bounds can be computed from maximal cliques and the minimum degree of induced subgraphs. A computational analysis shows that the treewidth of several graphs can be identified by these methods. For other graphs, however, more sophisticated techniques are necessary.

1 Introduction

In the 1980s and early 1990s, Robertson and Seymour defined the graph parameters *pathwidth* [28], *treewidth* [29], and *branchwidth* [30, 31] as well as the associated graph structures *path decomposition*, *tree decomposition*, and *branch decomposition*. Originally these concepts were introduced in the context of their fundamental research on graph minors. In addition, the

[†]Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, D-14195 Berlin, Germany. E-Mail: koster@zib.de. Corresponding author. The research was partly carried out during a visit of Utrecht University.

[‡]Institute of Information and Computing Sciences, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands. This author was partially supported by EC contract IST-1999-14186: Project ALCOM-FT (Algorithms and Complexity - Future Technologies).

[§]Department of Quantitative Economics, Universiteit Maastricht, P.O.Box 616, 6200 MD Maastricht, The Netherlands.

Keywords: treewidth, heuristics, lower bounds, computations

Mathematics Subject Classification (MSC 2000): 90C35, 05C85, 68W25

notions have proved to be of importance in computational complexity theory. Many $\mathcal{NP}$-hard graph problems can be solved in polynomial time for graphs with path-, tree-, or branchwidth bounded by a constant. Among these problems are many well-known combinatorial optimization problems such as the maximum independent set problem, the Hamiltonian cycle problem, and the Steiner tree problem. Moreover, several $\mathcal{PSPACE}$-complete problems can be solved in linear time for input restricted to graphs of bounded treewidth (cf. [6]).

All these results have been considered primarily of theoretical interest only. In recent years, a number of computational studies has shown that the results can also be applied successfully. A first attempt to benefit from the polynomial time solvability has been undertaken by Cook and Seymour [10]. They use branch decompositions to obtain close-to-optimal solutions of the traveling salesman problem. Path decompositions are used by Verweij [37] to solve lifting problems of cycle inequalities for the independent set problem. In [22, 23], tree decompositions are used to obtain lower bounds and optimal solutions for a special type of frequency assignment problems. Finally, tree decompositions are used to solve problems in the area of expert systems. The currently most efficient algorithm for the inference calculation in probabilistic (or Bayesian) networks builds upon a tree decomposition of a network's moralized graph [20, 24]. All these studies show that (dynamic programming) algorithms based on a path/tree/branch decomposition of the graph can be an alternative for integer programming techniques to solve hard (combinatorial) optimization problems.

The procedure to solve an optimization problem with for instance bounded treewidth involves two steps: (i) computation of a (good) tree decomposition, and (ii) application of an algorithm that solves instances of bounded treewidth in polynomial time (typically by a dynamic programming algorithm based on the tree decomposition). Whereas the second step is application-dependent, the calculation of a good tree decomposition of a graph can be done independently of the application. The need for a tree decomposition with optimal, or second best, good width becomes clear by the time and space requirements of the algorithm in the second step, which typically are exponentially in the width. A similar procedure is due for pathwidth and branchwidth. Henceforth, we focus on the computation of the treewidth of graphs from a practical point of view. Hicks [19] recently studied the practical computation of branchwidth for (specific classes of) graphs.

The treewidth of a graph is, in general, $\mathcal{NP}$-hard to compute. Much research has been carried out to determine the treewidth (or related notions) for classes of graphs. Moreover, for fixed k, linear time algorithms to determine whether a graph has treewidth k have been developed. Their use in practice, however, is very limited since the running time contains a large constant with k in the exponent. For small graphs, Shoikhet and Geiger [33] performed various experiments on randomly generated graphs with an adaption of an algorithm of Arnborg et al. [3]. Approximation with an $\mathcal{O}(\log n)$ approximation ratio can be done in polynomial time (here n denotes the number of vertices in the graph); finding a polynomial time constant approximation ratio algorithm remains open. We refer to [6] for a survey on these and other complexity issues for treewidth.

Heuristics without theoretical quality guarantee and easily computable lower bounds emerge as a practical alternative. Motivated by problems from combinatorial optimization and probabilistic networks, we describe a computational analysis of four heuristics and two lower bounds in this paper. We focus on algorithms that are not exponential in the treewidth (like the algorithm

of Becker and Geiger [5]). The heuristics are described in Section 3, whereas Section 4 is devoted to the lower bounds. The computational analysis of the heuristics and lower bounds on several classes of graphs is the topic of Section 5. The used graphs originate from frequency assignment problems, vertex coloring problems, and probabilistic networks. The paper is closed by concluding remarks and directions for further research.

2 Definitions and Notation

We briefly introduce the most important definitions and notation. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. Let $n = |V|$ and $m = |E|$ throughout this paper. The set of vertices adjacent to a vertex $v \in V$ is denoted by $N(v) = \{w \in V : vw \in E\}$. Let $\delta(v) := |N(v)|$ be the degree of v. A set of vertices $Q \subseteq V$ is called a *clique* in G if there is an edge between every pair of distinct vertices from Q. The cardinality $|Q|$ of Q, is the *size* of the clique. For a set W of vertices, the subgraph induced by W is the graph $G[W] := (W, E[W])$, with $E[W] := (W \times W) \cap E$. A graph H is a minor of a graph G if H can be obtained from G by vertex deletions, edge deletions, and edge contractions (the operation that replaces two adjacent vertices v and w by a single vertex that is connected to all neighbors of v and w).

The *tree decomposition* concept and its measure of value *treewidth* was introduced by Robertson and Seymour [29] in the context of graph minor research.

Definition 2.1 (Robertson and Seymour [29]) *Let $G = (V, E)$ be a graph. A **tree decomposition** of G is a pair $(T, \mathcal{X})$, where $T = (I, F)$ is a tree with node set I and edge set F, and $\mathcal{X} = \{X_i : i \in I\}$ is a family of subsets of V, one for each node of T, such that*

(i). $\bigcup_{i \in I} X_i = V$,

(ii). for every edge $vw \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$, and

(iii). for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

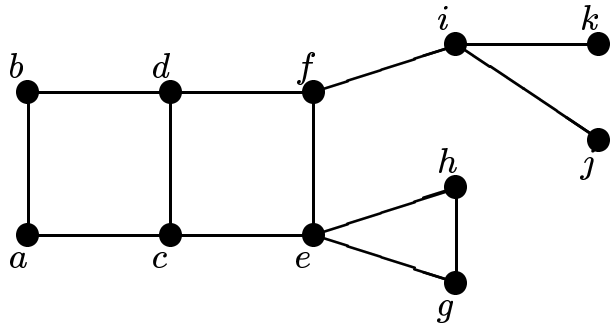
*The **width** of a tree decomposition is $\max_{i \in I} |X_i| - 1$. The **treewidth** of a graph G, denoted by $tw(G)$, is the minimum width over all possible tree decompositions of G.*

The third condition of the tree decomposition is equivalent to the condition that for all $v \in V$, the set of nodes $\{i \in I : v \in X_i\}$ is connected in T, i.e., they form a subtree. Figure 1 shows an example of a graph together with an optimal tree decomposition.

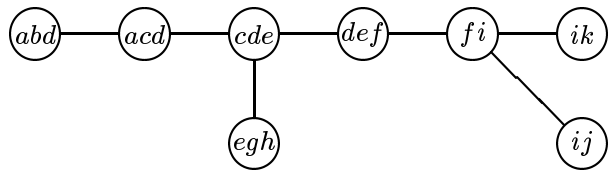
If we restrict the tree T to be a path, a *path decomposition* of G is obtained. The minimum width over all possible path decompositions of G is called the *pathwidth* of G. Many definitions equivalent to treewidth or pathwidth are known, see [7] for an overview. For the heuristics described in the next section, two of these alternative characterizations are exploited.

3 Heuristics

In this section, we present four heuristics for computing the treewidth of a graph. The first three are based on algorithms for determining whether a graph is triangulated. The fourth one



(a) graph



(b) tree decomposition with width 2

Figure 1: Example of a graph with a tree decomposition; the subsets X_i are displayed at the nodes

bases on minimal separating vertex sets.

3.1 Graph Triangulation Heuristics

A graph is called *triangulated* if every cycle of length at least four contains a chord, that is, two non-consecutive vertices on the cycle are adjacent. Triangulated graphs are also called *chordal* due to the existence of a chord in every cycle. Triangulated graphs are among the first classes of graphs for which it has been proved that they are perfect.

The reason for introducing triangulated graphs in this paper is the following property of these graphs:

Theorem 3.1 (Gavril [16]) *A graph $G = (V, E)$ is triangulated if and only if G can be constructed as the intersection graph of subtrees of trees, i.e., there exists a tree $T = (I, F)$ such that one can associate a subtree $T_v = (I_v, F_v)$ of T with each vertex $v \in V$, such that $vw \in E$ if and only if $I_v \cap I_w \neq \emptyset$.*

Although the concept of tree decomposition was not yet introduced by the time Gavril formulated Theorem 3.1, exactly this concept was used to characterize triangulated graphs. To state the result in a different way, a graph is triangulated if and only if there exists a tree decomposition with the additional property that $vw \in E$ if and only if $I_v \cap I_w \neq \emptyset$. This additional property

guarantees that the tree decomposition has minimal width. Let $X_i := \{v \in V : i \in I_v\}$. Non-adjacent vertices are not in a common subset X_i. Hence, a maximum cardinality subset X_i contains the vertices of a maximum cardinality clique Q in G. Since it also holds that in any tree decomposition $(T, \mathcal{X})$ and every maximum clique Q, there exists a node $i \in I$ with $Q \subseteq X_i$ (cf. Section 4.1), the tree decomposition has minimum width. Moreover, it follows that the treewidth of a triangulated graph equals the maximum clique number minus one, $tw(G) = \omega(G) - 1$.

Lemma 3.2 *For every graph $G = (V, E)$, there exists a triangulation of G, $\bar{G} = (V, E \cup E_t)$, with $tw(\bar{G}) = tw(G)$.*

Proof. Let G be a general graph and $(T, \mathcal{X})$ a tree decomposition of minimum width. We construct a graph $\bar{G} = (V, \bar{E})$ by the following rule: $vw \in \bar{E}$ if and only if $vw \in X_i$ for some $i \in I$. From Theorem 3.1 it is clear that $\bar{G}$ is triangulated. From the second condition of a tree decomposition, the edge set $\bar{E}$ can be divided into two parts E and E_t. So, $\bar{G}$ is a triangulation of G by the addition of the *triangulation edges* E_t. Moreover, the treewidth of G and $\bar{G}$ is equal, $tw(\bar{G}) = tw(G)$. ■

Corollary 3.3 *Finding the treewidth of a graph G is equivalent to finding a triangulation $\bar{G}$ of G with minimum clique size.*

Since finding the treewidth of a graph is $\mathcal{NP}$-hard, also finding a triangulation with minimum clique number is an $\mathcal{NP}$-hard problem. However, the clique number (minus one) of any triangulation $\bar{G}$ of a graph G provides an upper bound on the treewidth. Moreover, $\omega(\bar{G})$ can be computed in polynomial time. Therefore, we are looking for algorithms that triangulate a graph, and by that provide upper bounds for the treewidth. There exist several algorithms to test whether a graph is triangulated. These recognition algorithms either detect that the graph is triangulated or return a triangulation of the graph. In the subsequent subsections, three of these algorithms are presented, two variants of the lexicographic breadth first search algorithm of Rose et al. [32], and the maximum cardinality search algorithm of Tarjan and Yannakakis [35]. All three algorithms are based on the characterization of triangulated graphs by a *perfect elimination scheme*.

A vertex $v \in V$ is called *simplicial* if its neighbors $N(v)$ induce a complete subgraph of G. An ordering $\sigma = [v_1, \ldots, v_n]$ of the vertices is called a *perfect elimination scheme* if v_i is a simplicial vertex of the induced subgraph $G[\{v_i, \ldots, v_n\}]$ for all $i = 1, \ldots, n$. So, G contains a simplicial vertex v_1, and every time a simplicial vertex v_i is removed from the graph, a new simplicial vertex v_{i+1} exists. Fulkerson and Gross [15] proved that a graph is triangulated if and only if it has a perfect elimination scheme. Given a perfect elimination scheme, the maximal cliques of G are of the form $\{u\} \cup M(v)$, where $M(v) = \{w \in N(v) : \sigma^{-1}(v) < \sigma^{-1}(w)\}$ denotes the set of higher ordered neighbors. The value $\max_{v \in V} |M(v)|$ is also called the *dimension* of G. Arnborg [2] proved that the dimension of G equals the treewidth.

For a non-triangulated graph G and an elimination scheme σ, a triangulation $\bar{G}$ can be constructed by Algorithm 1. Moreover, the algorithm returns the treewidth of $\bar{G}$, which is an upper bound for the treewidth of G. The subset $E_t := \bar{E} \setminus E$ is also called the *fill-in* of G.

Algorithm 1 Triangulation of a graph, given an elimination scheme

Input: connected graph $G = (V, E)$, elimination scheme $\sigma = [v_1, \ldots, v_n]$
Output: triangulated graph $\bar{G} = (V, \bar{E})$ with $tw(\bar{G}) \geq tw(G)$

```
for v ∈ V do
    M(v) := N(v)
end for
Ē := E, tw := 0
for i = 1 to n do
    if |M(v_i)| > tw then
        tw := |M(v_i)|                          // update treewidth
    end if
    for u, w ∈ M(v_i) and uw ∉ Ē do
        Ē := Ē ∪ {uw}                           // add edge between higher ordered neighbors
        M(u) := M(u) ∪ {w}, M(w) := M(w) ∪ {u}  // update adjacency lists
    end for
    for u ∈ M(v_i) do
        M(u) := M(u) \ {v_i}                    // v_i is lower ordered than u
    end for
end for
```

The easiest way to recognize triangulated graphs is the construction of an elimination scheme σ, which is perfect (i.e., no edges are added by Algorithm 1) in case the graph is indeed triangulated. The following observation plays an important role in such a recognition algorithm.

Proposition 3.4 (Fulkerson and Gross [15]) *Let G be a triangulated graph. Any simplicial vertex can start a perfect elimination scheme for G.*

So an algorithm could be: search for a simplicial vertex in the graph, add it to the elimination scheme and remove it from the graph. If at some point no simplicial vertex exists, the graph is not triangulated, otherwise the graph indeed is. Such an algorithm requires n searches for a simplicial vertex, which requires an adjacency-test for all pairs of neighbors of a vertex. More efficient algorithms can be constructed by using another observation.

Proposition 3.5 (Dirac [12]) *Every triangulated, non-complete graph has two non-adjacent simplicial vertices.*

This observation (combined with Proposition 3.4) implies that at every step of the above described algorithm we have the choice between at least two non-adjacent vertices to place at the next position of the elimination scheme. Hence, without loss of generality, we can select a vertex v_n for the last position in the elimination scheme. Next, we choose a vertex v_{n-1} adjacent to v_n for the $(n-1)$th position. Further vertices have to be selected in such a way that their already selected neighbors form a clique. In this way, the elimination scheme is constructed backwards. Since, vertices need not be identified as simplicial explicitly, algorithms that construct the elimination scheme are more efficient compared with forward construction algorithms. In the next two subsections, three such backward algorithms are presented.

For non-triangulated graphs these algorithms can be applied to obtain a triangulation, and thus, an upper bound on the treewidth of G. Since in general G is not triangulated, the choice of the vertex v_n for the last position of the elimination scheme influences the upper bound returned by the algorithm.

3.1.1 Lexicographic Breadth First Search

In this subsection, we present two variants of what is called a lexicographic breadth first search algorithm. These algorithms as well as the maximum cardinality search algorithm in the next subsection only differ in the way the vertices for positions $n-1, n-2, \ldots, 1$ are determined. All three algorithms guarantee that the elimination scheme is perfect if and only if G is triangulated. Hence, for a triangulated graph G, the vertices have to be ordered in such a way that for all i, $1 \leq i \leq n$ it should hold that v_i is simplicial in $G[\{v_i, \ldots, v_n\}]$.

The algorithm presented in Rose et al. [32] is known as the *lexicographic breadth-first search* recognition algorithm (see also [17, Section 4.3]). A first version of this algorithm focuses primarily on the perfectness of the elimination scheme (referred to as `LEX_P`). In the original version, it terminates as soon as it recognizes that no perfect elimination scheme exists, i.e., as soon as no vertex v_i can be found that is simplicial in $G[\{v_i, \ldots, v_n\}]$. Leaving out this termination of the algorithm, it returns an elimination scheme that can be used to find a triangulation of G. The algorithm labels the vertices with their already ordered neighbors. Each label consists of the positions of the already ordered neighbors in decreasing order. The unordered vertex with the lexicographic highest label is selected in every iteration (cf. Algorithm 2). The most efficient implementation of Algorithm 2 requires $\mathcal{O}(n+m')$ time and space, where $m' = m + |E_t|$ (see [32] for details). If we start the algorithm n times with every time another vertex v^*, the overall running time is $\mathcal{O}(n^2 + nm')$.

Algorithm 2 Lexicographic Breadth First Search, variant Perfect (`LEX_P`)

Input: connected graph $G = (V, E)$, vertex $v^* \in V$
Output: triangulated graph $G_T = (V, E \cup E_t)$ with $tw(G_T) \geq tw(G)$; if G is triangulated, then $E_t \equiv \emptyset$

```
E_t := ∅
for v ∈ V do
   label(v) := ∅                          // set of numbers listed in decreasing order
end for
S := V                                    // set of unordered vertices
for i = |V| to 1 do
   if i ≡ |V| then
      u := v*
   else
      u := arg max_{v∈S} label(v)         // u is a vertex with lexicographic largest label
   end if
   σ(i) := u                              // add u to ordering at position i
   for j, k ∈ {i+1, ..., |V|}, j ≠ k do
      if {σ(i), σ(j)}, {σ(i), σ(k)} ∈ E ∧ {σ(j), σ(k)} ∉ E ∪ E_t then
         E_t := E_t ∪ {σ(j), σ(k)}        // add edge to triangulation
      end if
   end for
   S := S \ {u}
   for w ∈ N(u) ∩ S do
      label(w) := label(w) ∪ {i}          // add i to the label of all unordered neighbors
   end for
end for
```

Compared with the `LEX_P` algorithm, the second version of the lexicographic breadth first search algorithm has the feature that a minimal triangulation of G is returned (henceforth called `LEX_M`). A triangulation is called *minimal* if the removal of any edge e of the fill-in E_t results in a non-triangulated graph. To guarantee this property of the triangulation, not only the neighbors of an ordered vertex are labeled, but also all of the vertices that can be reached by a path of unordered

vertices with lower labels (cf. Algorithm 3). Due to this extension of the labeling procedure, the running time of the algorithm increases to $\mathcal{O}(nm')$, whereas the space requirements $\mathcal{O}(n+m')$ are equal to those of `LEX_P`. Running `LEX_M` once for all $v^* \in V$ results in an overall running time of $\mathcal{O}(n^2 m')$.

Algorithm 3 Lexicographic Breadth First Search, variant Minimal (`LEX_M`)

Input: connected graph $G = (V, E)$, vertex $v^* \in V$
Output: minimal triangulated graph $G_T = (V, E \cup E_t)$ with $tw(G_T) \geq tw(G)$; if G is triangulated, then $E_t \equiv \emptyset$

```
E_t := ∅
for v ∈ V do
   label(v) := ∅                                  // set of numbers listed in decreasing order
end for
S := V                                            // set of unordered vertices
for i = |V| to 1 do
   if i ≡ |V| then
      u := v*
   else
      u := arg max_{v∈S} label(v)                 // u is a vertex with lexicographic largest label
   end if
   σ(i) := u                                      // add u to ordering at position i
   for j, k ∈ {i+1, ..., |V|}, j ≠ k do
      if {σ(i), σ(j)}, {σ(i), σ(k)} ∈ E ∧ {σ(j), σ(k)} ∉ E ∪ E_t then
         E_t := E_t ∪ {σ(j), σ(k)}                // add edge to triangulation
      end if
   end for
   S := S \ {u}
   for w ∈ S : ∃ path {u = v_1, ..., v_{k+1} = w} in G
         with v_j ∈ S and label(v_j) < label(w) for j = 2, 3, ..., k do
      label(w) := label(w) ∪ {i}                  // add i to the label of all unordered neighbors
   end for
end for
```

3.1.2 Maximum Cardinality Search

The Maximum Cardinality Search (`MCS`) recognition algorithm of Tarjan and Yannakakis [35] uses another criterion to select the vertices. Instead of selecting the vertex with the highest lexicographic label, they select the vertex which is adjacent to the highest number of ordered vertices. The algorithm does not guarantee a minimal triangulation and runs in $\mathcal{O}(n+m')$ time.

3.2 Minimum Separating Vertex Set Heuristic

The fourth heuristic described in this paper was developed in the context of solving frequency assignment problems with a tree decomposition approach [21]. It bases on another characterization of tree decompositions. Every tree decomposition can be transformed to a tree decomposition in which the vertex set associated to an internal node separates the graph into at least two components, i.e., the vertices associated with the node form a *separating vertex set*. The heuristic therefore searches for separating vertex sets. To find a good tree decomposition, we in fact search for *minimum* separating vertex sets. This explains the name of the heuristic. The choice for minimum separating vertex sets is motivated by the observation of Dirac [12] that a graph G is triangulated if and only if every minimal separating vertex set induces a complete subgraph of

Algorithm 4 Maximum Cardinality Search (`MCS`)

Input: connected graph $G = (V, E)$, vertex $v^* \in V$
Output: triangulated graph $G_T = (V, E \cup E_t)$ with $tw(G_T) \geq tw(G)$; if G is triangulated, then $E_t \equiv \emptyset$

```
E_t := ∅
for v ∈ V do
   counter(v) := 0                          // counts the number of ordered neighbors
end for
S := V                                      // set of unordered vertices
for i = |V| to 1 do
   u := arg max_{v∈S} counter(v)            // vertex with largest number of ordered neighbors
   σ(i) := u                                // add u to ordering at position i
   for j, k ∈ {i+1, ..., |V|}, j ≠ k do
      if {σ(i), σ(j)}, {σ(i), σ(k)} ∈ E ∧ {σ(j), σ(k)} ∉ E ∪ E_t then
         E_t := E_t ∪ {σ(j), σ(k)}          // add edge to triangulation
      end if
   end for
   S := S \ {u}
   for w ∈ N(u) ∩ S do
      counter(w) := counter(w) + 1          // increase counter for all unordered neighbors
   end for
end for
```

G. So, for a triangulation $\bar{G}$ of G with $tw(\bar{G}) = tw(G)$, the vertex sets associated with internal nodes of T correspond to minimum vertex separating sets. We are looking for exactly those sets in the original graph G. Before describing the minimum separating vertex set (`MSVS`) heuristic, we briefly describe how a separating vertex set of minimum cardinality can be found in a graph.

3.2.1 Minimum Separating Vertex Sets in Graphs

To find a minimum separating vertex set S in a graph $G = (V, E)$, we have to compute the minimum st-separating set for all non-adjacent $s, t \in V$.

Definition 3.6 *An st-separating set of a graph $G = (V, E)$ is a set $S \subseteq V \setminus \{s, t\}$ with the property that any path from s to t passes through a vertex of S. The minimum separating vertex set of G is given by the st-separating set S with minimum cardinality over all combinations $st \notin E$.*

The st-separating set with minimum cardinality can be found efficiently using Menger's theorem (see also Ahuja, Magnanti, and Orlin [1]).

Theorem 3.7 (Menger [26]) *Given a graph $G = (V, E)$ and two distinct non-adjacent vertices $s, t \in V$, the minimum number of vertices in an st-separating set is equal to the maximum number of vertex-disjoint paths connecting s and t.*

So, we have to calculate the maximum number of vertex-disjoint paths. This problem is solvable in polynomial time by standard network flow techniques. First, G is transformed into a directed graph $D = (V, A)$, in which each edge vw is replaced by two arcs (v, w) and (w, v). Next, we construct an auxiliary directed graph D', with weights on the arcs, by

- replacing each vertex v by two vertices v' and v'',

- redirecting each arc with head v to v', and introducing a weight of ∞,
- redirecting each arc with tail v to v'', and introducing a weight of ∞, and
- adding an arc from v' to v'' with weight 1.

Then, the minimum number of vertices in an st-separating set in G is equal to the minimum weight of an $s'' - t'$ cut in D'. So, if we calculate the minimum $s'' - t'$ cut for every non-adjacent pair $s, t \in V$, we obtain the minimum separating vertex set. Note that since the graph D' is a *directed* graph, we have to solve $\mathcal{O}(n^2)$ minimum cut problems. In other words, we cannot use the algorithm of Gomory and Hu [18], which solves the all pairs minimum cut problem for *undirected* graphs by solving only $\mathcal{O}(n)$ minimum cut problems.

3.2.2 MSVS Heuristic

The `MSVS` heuristic is in fact an improvement heuristic, i.e., it starts with a tree decomposition and tries to improve its width. The heuristic becomes a constructive heuristic if it starts with the trivial tree decomposition $(T, \mathcal{X})$: $|I| = 1$, $F = \emptyset$, and $X_i = V$.

We assume that the tree decomposition $(T, \mathcal{X})$ to start with is *minimal* in the sense that no $v \in V$ can be removed from a subset X_i, $i \in I$, without violating one of the tree decomposition conditions. Suppose there exists an unique node with maximum cardinality associated vertex set. Then, a replacement of this node by a set of nodes, all with smaller associated vertex sets, reduces the width of the tree decomposition. If the node is not unique, then we have to replace all nodes with maximum cardinality vertex set by nodes with smaller associated vertex sets to achieve an improvement in the width.

For each replacement of a node by new nodes, the new nodes have to be connected with the remaining parts of the tree in such a way that again all conditions of a tree decomposition are satisfied. Let $i \in I$ be a node with maximum cardinality vertex set. Suppose we would like to replace i by a new node i^* and q nodes I^* as displayed in Figure 2. So, all new nodes $p \in I^*$ are connected with i^*, some of the previous neighbors $j \in N(i)$ are connected with i^*, and all other neighbors are connected with i^* via a (unique) node $p \in I^*$. Here, we allow that multiple neighbors $j \in N(i)$ are connected with one and the same new node p. But we also allow that not every new node $p \in I^*$ has to be connected with nodes $j \in N(i)$.

We require that the vertex set X_{i^*} of node i^* defines a minimum separating vertex set in a graph $H = (V(H), E(H))$ to be specified in the following. Each of the $q \geq 2$ nodes $p \in I^*$ corresponds with a component of $H[V(H) \setminus X_{i^*}]$. Let Y_p be the vertex set of component p. Note that each $v \in X_{i^*}$ is connected with all components, since otherwise X_{i^*} is not a minimum separating vertex set. As a consequence, for every $p \in I^*$, the associated vertex set has to consist of both the vertices in the component and the separating vertex set, $X_p := Y_p \cup X_{i^*}$ (otherwise the condition that for all $vw \in E$ there exist a $i \in I$ with $v, w \in X_i$ cannot be guaranteed afterwards). Now, the graph H can be specified.

Lemma 3.8 *To guarantee that the newly constructed pair $(T, \mathcal{X})$ satisfies all conditions of tree decomposition, the vertex set and edge set of graph $H = (V(H), E(H))$ have to satisfy $X_i \subseteq V(H)$ and $\bigcup_{j \in N(i)} Q(X_i \cap X_j) \cup E_G[X_i] \subseteq E(H)$, respectively. These sets are also sufficient.*

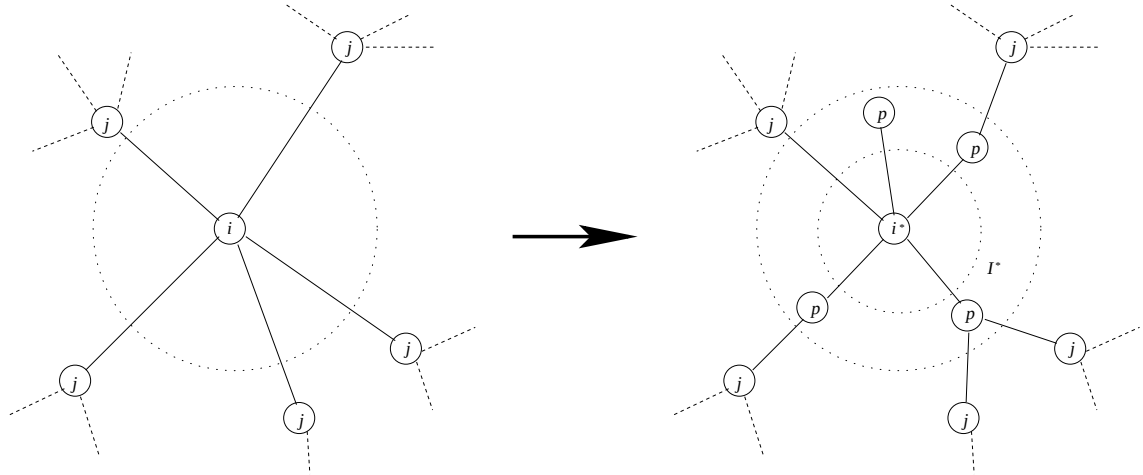


Figure 2: Improvement step of a tree decomposition

Proof. Since we assume that the algorithm starts with a minimal tree decomposition, no vertex $v \in X_i$ can be left out without violating the tree decomposition conditions. As a consequence, every vertex $v \in X_i$ has to be in the vertex set of H, otherwise the condition cannot be guaranteed afterwards anymore.

For the edge set we distinguish between pairs of vertices $v, w \in X_j$ for some $j \in N(i)$ and $v, w \notin X_j$ for all $j \in N(i)$. First, let there be a $j \in N(i)$ with $v, w \in X_j$. Let $p(j)$ be the new node $p \in I^*$ between j and i^* if such a node exists, otherwise let $p(j) = i^*$. By $v \in X_j$ and the definition of the subset $X_{p(j)}$, we have $v \in X_{p(j)}$. The same holds for w. Thus v and w have to be in the same component of the graph $H[X_i \setminus X_{i^*}]$, which only can be guaranteed by $vw \in E(H)$. Hence, $Q(X_i \cap X_j) \subset E(H)$ for all $j \in N(i)$.

Now, let $v, w \in X_i$ with $v, w \notin X_j$ for all $j \in N(i)$. So, i is the unique node with $vw \in X_i$. If $vw \in E$, this node is required to guarantee the second condition of a tree decomposition. To guarantee that after the replacement of i, there still is a node with both v and w belonging to the associated vertex set, v and w should not end up in two different components. Thus, $vw \in E(H)$. If $vw \notin E$, then separation of v and w does not violate any condition for the resulting tree decomposition. Hence, it is not necessary to include non-adjacent v and w in $E(H)$. ■

Lemma 3.9 *The cardinality of the associated vertex sets decreases,* $\max\{|X_{i^*}|, \max_{p \in I^*} |X_p|\} < |X_i|$*, if and only if* H *is not a complete graph.*

Proof. If H is complete, then no minimum separating vertex set exists, and thus no reduction in the cardinality of the associated vertex sets can be achieved. If H is non-complete, then there exist $v, w \in V(H)$ with $vw \notin E(H)$. The minimum vw-separating vertex set S contains at least one, and at most $|V(H)| - 2$ vertices. Every component of $H[V(H) \setminus S]$ contains also at least one vertex. Hence, $\max_{p \in I^*} |X_p| < |X_i|$. ■

From Lemma 3.9, it directly follows that as long as H is non-complete for all nodes $i \in I$ with

maximum cardinality associated vertex set, improvements on the treewidth are achieved. Note that by construction of H and Y_p, the set X_j has a non-empty intersection with at most one set Y_p, $p \in \{1, \ldots, q\}$. Let $v, w \in X_i \cap X_j$, then $vw \in Q(X_i \cap X_j) \subset E(H)$, which implies that v and w cannot be separated by X_{i^*}. So, either $v, w \in X_{i^*}$ or $v, w \in Y_p \cup X_{i^*} =: X_p$ for only one $j \in \{1, \ldots, m\}$. Therefore, we connect each neighbor $j \in N(i)$ with the node $p \in \{1, \ldots, q\}$ for which the intersection of X_j and Y_p is non-empty, or in case there is no such p we connect j with i^* (cf. Figure 2). Algorithm 5 shows a pseudo-code implementation of the heuristic.

Algorithm 5 Minimum Separating Vertex Sets (`MSVS`)

Input: initial tree decomposition $(T, \mathcal{X})$
Output: modified tree decomposition $(\bar{T}, \bar{\mathcal{X}})$ with $tw((\bar{T}, \bar{\mathcal{X}})) \leq tw((T, \mathcal{X}))$
// if for a maximal subset X_i, H is non-complete, the width can be improved
while $\exists i \in I : |X_i| \equiv \max_{j \in I} |X_j|$ and $H = (X_i, E(H))$ non-complete
  with $E(H) = \bigcup_{j \in N(i)} Q(X_i \cap X_j) \cup E_G[X_i]$ **do**
  $N_{old}(i) := N(i)$ *// store old neighbors of i*
  $F := F \setminus \{ij : j \in N_{old}(i)\}$ *// disconnect i from tree*
  let $S \subset X_i$ be a minimum separating vertex set in H (cf. Section 3.2.1)
  let q be the number of components of $H[X_i \setminus S]$
  $n := |I|$ *// current number of nodes*
  $I := I \cup \{n+1, \ldots, n+q\}$ *// construct q new nodes*
  $F := F \cup \{ij : j = n+1, \ldots, n+q\}$ *// connect new nodes with i*
  for $p = 1$ to q **do**
    let $Y_p \subset X_i$ be the set of vertices in component p of $H[X_i \setminus S]$
    $X_{n+p} := Y_p \cup S$ *// define new vertex subsets*
  end for
  $X_i := S$
  for $j \in N_{old}(i)$ **do**
    if $\exists p \in \{1, \ldots, q\}$ with $X_j \cap Y_p \neq \emptyset$ **then**
      $F := F \cup \{n+p, j\}$ *// reconnect old neighbors*
    else
      $F := F \cup \{i, j\}$ *// reconnect old neighbors*
    end if
  end for
end while

The width of the resulting tree decomposition approximates the minimal treewidth. Note that as long as the separating vertex sets S form cliques in the original graph G, the algorithm operates in an exact way, since the optimal tree decomposition will contain a node for every clique that separates the graph in multiple components.

4 Lower Bounds

By applying the heuristics on a set of graphs, a reciprocative comparison of the algorithms can be carried out. The results, however, do not tell us anything about the quality of the best heuristic. For that purpose, we need to know the treewidth of the graphs, which requires to solve an $\mathcal{NP}$-hard problem. Alternatively, lower bounds on the treewidth can be used to benchmark the heuristics. In this section, two lower bounds that are used in this computational study are presented.

4.1 Maximum Clique Bound

It is well known that the treewidth of a graph does not increase under taking minors. So, the treewidth of a minor H of G is a lower bound for the treewidth of G, $tw(H) \leq tw(G)$. The minor of G that provides the best lower bound is of course G itself, or minors H with $tw(H) = tw(G)$. For general minors, computing treewidth is however as difficult as computing the treewidth of G.

Since every graph induced by a clique $Q \subseteq V$ in G is a minor of G, $tw(G) \geq |Q| - 1$. The best bound derived this way is for the maximum clique. Hence, the maximum clique number $\omega(G)$ of a graph G minus one is a lower bound for the treewidth. Computing the maximum clique number is $\mathcal{NP}$-hard, in general. Nevertheless, several studies have shown that for not too large and not too dense graphs, the maximum clique number can be computed within reasonable time [4, 27].

4.2 Maximum Minimum Degree Bound

A trivial lower bound on the treewidth is given by the minimum degree of the graph: let $(T, \mathcal{X})$ be minimal tree decomposition with optimal width k, and let $i \in I$ be a leaf node with predecessor j. By minimality of the tree decomposition, there is a vertex $v \in X_i \setminus X_j$. Hence, $\delta(v) \leq k$.

The above implies that the minimum degree of a minor provides a lower bound as well. Since every (induced) subgraph of G is a minor of G, the maximum of the minimum degree over all subgraphs, $MMD(G) = \min_{H \subseteq G} \delta(H)$, bounds the treewidth from below. This so-called *maximum minimum degree* bound can be computed in polynomial time.

Lemma 4.1 *The maximum minimum degree of G is $MMD(G) = \max\{\delta(v), MMD(G[V \setminus \{v\}])\}$ for all $v \in V$ of minimum degree.*

Proof. Let $v \in V$ be a vertex in G of minimum degree. The minimum degree of all subgraphs G' containing v is at most the minimum degree of G. Thus, either $MMD(G) = \delta(v)$, or the maximum minimum degree bound equals the same bound for the graph without v. ■

This result directly provides an algorithm to compute the maximum minimum degree: remove sequentially minimum degree vertices of the graph and keep track of the maximum of these minimum degrees, see Algorithm 6.

Since the maximum clique of G is a subgraph, $MMD(G) \geq \omega(G) - 1$. In fact, Szekeres and Wilf [34] proved that $\chi(G) - 1 \leq MMD(G)$. So, a bound that is at least as good as the maximum clique bound and the minimum coloring bound can be found in polynomial time. Using a Van Embe Boas-tree [36], the algorithm can be implemented in $\mathcal{O}(n \log\log n + m)$ time.

Note that the maximum minimum degree $MMD(G)$ dualizes with another graph theoretic concept called *k-degeneracy* [25]. A graph is called k-degenerate if and only if every induced subgraph of G has a vertex of degree at most k. The minimum k for which G is k-degenerate equals $\texttt{MMD}(G)$.

Algorithm 6 Maximum Minimum Degree (`MMD`)

Input: connected graph $G = (V, E)$
Output: $MMD \leq tw(G)$

```
for v ∈ V do
    n(v) := N(v)
end for
S := V, MMD := 0.                          // S defines induced subgraph
while S ≠ ∅ do
    u := arg min_{v∈S} |n(v)|              // vertex with minimum degree in subgraph
    if |n(u)| > MMD then
        MMD := |n(u)|                      // update maximum
    end if
    for v ∈ n(u) do
        n(v) := n(v) \ {u}
    end for
    S := S \ {u}                           // reduce subset
end while
```

5 Computational Analysis

The algorithms described in the previous sections are tested on graphs obtained from applications like vertex coloring, frequency assignment, and probabilistic networks. All instances are well known in the application area, and are often used for benchmarking newly proposed methodologies. All algorithms have been implemented in C++ on a Linux PC with Pentium III 800 MHz processor. Consecutively, we discuss the results for instances from frequency assignment, probabilistic networks, and vertex coloring.

5.1 Frequency Assignment

The frequency assignment problem deals with the allocation of frequencies to the transmitters of a wireless communication network. Interference between signals transmitted on (almost) the same frequency restricts the assignment of the frequencies. Moreover, certain frequencies can be locally blocked at a transmitter, i.e., it is prohibited to assign the frequency to the transmitter. Depending on the application at hand, we want to minimize the number of used frequencies, the span of the used frequencies, the blocking probability of the network, or the total interference in the network. For more information on frequency assignment, we refer to [13, 21] or to FAP web [14].

In the context of the EUCLID CALMA project [9], instances for a variety of frequency assignment problems have been made available. We restrict ourselves to the instances where the objective is to minimize the overall interference in an assignment. In [21], it was shown that application of a tree decomposition approach yield to good lower bounds and optimal solutions for these instances. In total 11 instances are available with a total of 9 different graphs. Several preprocessing ideas (for frequency assignment) led to an additional set of 7 instances (3 instances were solved by preprocessing, for 1 the graph did not change). We tested the diverse algorithms on these 16 instances.

Table 1 shows statistics for the performance of the algorithms on these instances. The maximum clique size (`MC`) is computed with a standard tree search (branch-and-bound) algorithm. Several remarks can be made. The table shows that the `MMD` lower bound is indeed better than the clique

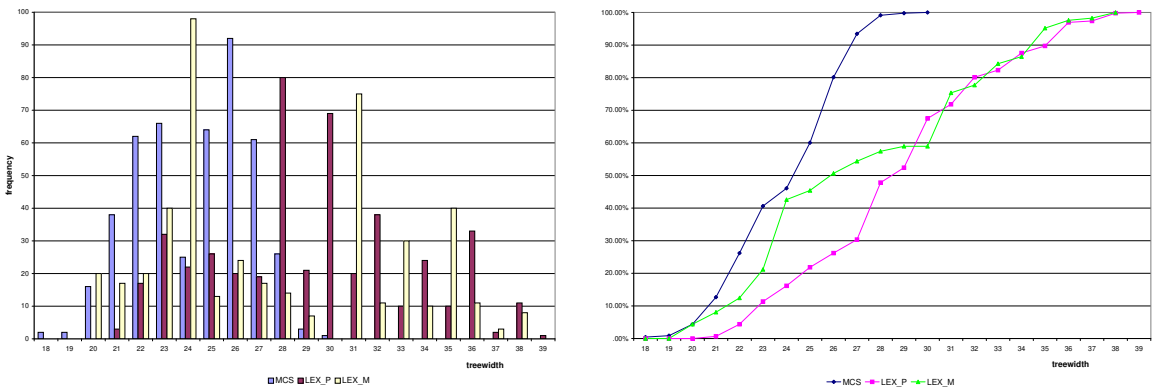
instance	$\|V\|$	$\|E\|$	MC	MMD	MCS			LEX_P			LEX_M			MSVS
					min.	avg.	max.	min.	avg.	max.	min.	avg.	max.	
CELAR06	100	350	10	10	11	11.27	14	11	13.69	14	11	13.61	14	11
CELAR07	200	817	10	11	18	22.25	26	19	23.64	32	17	20.64	29	17
CELAR08	458	1655	10	11	18	24.36	30	21	29.16	39	20	27.71	38	18
CELAR09	340	1130	10	11	18	25.09	31	21	29.39	39	17	27.82	38	18
GRAPH05	100	416	8	8	28	33.60	36	29	36.60	47	28	35.13	43	27
GRAPH06	200	843	8	8	54	61.94	74	61	72.06	83	59	70.93	85	56
GRAPH11	340	1425	7	7	102	115.43	127	106	121.88	143	106	120.80	139	102
GRAPH12	340	1255	5	5	97	108.31	121	99	114.77	128	99	113.73	128	93
GRAPH13	458	1877	6	6	136	148.89	162	147	161.22	178	147	160.55	176	133
CELAR06PP	82	327	10	10	11	11.33	14	11	13.65	16	11	13.52	14	11
CELAR07PP	162	764	10	11	18	21.60	26	19	25.65	30	17	25.62	29	17
CELAR08PP	365	1539	10	11	19	25.04	29	21	32.28	38	19	28.58	38	18
CELAR09PP	67	165	7	7	7	7.00	7	7	7.00	7	7	7.00	7	7
GRAPH06PP	119	348	5	5	19	23.85	30	21	26.97	33	20	25.83	31	18
GRAPH12PP	61	123	4	4	5	6.85	9	5	6.48	8	4	5.48	7	4
GRAPH13PP	456	1874	6	6	133	149.19	163	147	161.20	178	147	160.52	176	133

Table 1: Statistics for CALMA frequency assignment problems

instance	CPU time (in seconds)				
	MC	MCS	LEX_P	LEX_M	MSVS
CELAR06	0.11	0.07	0.09	0.74	9.30
CELAR07	0.44	0.48	0.57	5.90	101.62
CELAR08	1.49	5.02	6.08	46.24	603.19
CELAR09	0.74	2.55	2.39	28.17	580.30
GRAPH05	0.06	0.68	0.84	1.98	34.98
GRAPH06	0.20	15.42	20.08	28.88	323.19
GRAPH11	0.33	247.30	262.45	292.71	2290.80
GRAPH12	0.18	199.57	222.60	246.63	2124.58
GRAPH13	0.50	994.37	1118.32	1170.50	7507.91
CELAR06PP	0.09	0.04	0.06	0.52	5.67
CELAR07PP	0.39	0.36	0.46	4.08	58.65
CELAR08PP	1.24	2.51	4.26	28.12	412.07
CELAR09PP	0.00	0.03	0.03	0.10	0.03
GRAPH06PP	0.01	0.48	0.54	2.26	22.81
GRAPH12PP	0.00	0.02	0.01	0.16	2.04
GRAPH13PP	0.46	970.08	1092.34	1111.65	7602.44

Table 2: Computation times for CALMA frequency assignment problems

bound for some instances. The MSVS heuristic produces the best result for 14 out of 16 instances, MCS 7 times, LEX_P 3 times, and LEX_M 7 times. In Table 1, not only the upper bounds computed by the MCS, LEX_P, and LEX_M heuristics are reported, but also the average and maximum bound computed in the runs with different vertex v^* (cf. Algorithms 2, 3, and 4). These values show that the bounds computed by LEX_P and LEX_M are spread out over a wider range of values than the bounds computed by MCS. For CELAR08, we depict this observation in Figure 3. The range



(a) histogram

(b) cumultative

Figure 3: Statistics triangulation heuristics for CELAR08

of the bounds is not only smaller than those for the other heuristics, but also smaller values are more often generated. Next, note that the treewidth for two instances (CELAR09PP and GRAPH12PP) is known since the best lower and upper bound are equal. For CELAR09PP, all algorithms produce the same bound, even for arbitrary starting vertex. On the other hand, the table shows (very) large gaps between lower and upper bounds for the larger GRAPH instances.

This remarkable difference between larger CELAR and GRAPH instances seems to be caused by the origin of the instances: the CELAR instances are taken from real-life whereas the GRAPH instances are computer generated. Finally, we would like to point out that the figures show that the frequency assignment preprocessing is very effective, also for reducing the treewidth of the graphs: the upper bound for the treewidth reduces from 93 to 4 for GRAPH12. Also for other instances, Table 1 shows substantial reductions.

Table 2 shows the computation times of the heuristics as well as the `MC` bound. The `MMD` bound is computed in a fraction of a second for all instances. The computation times of the heuristics `MCS`, `LEX_P`, and `LEX_M` do not differ that much. The `MSVS` heuristic consumes much more time than the other ones.

5.2 Probabilistic networks

We continue our computational study with 8 real-life probabilistic networks from the field of medicine. Table 3 shows the origin of the instances and the size of the network. The most efficient way to compute the inference in a probabilistic network is by the use of the *junction-tree propagation* algorithm of Lauritzen and Spiegelhalter [24]. This algorithm uses a tree decomposition of the network's moralized graph. The moralization of a network (or directed graph) $D = (V, A)$ is the undirected graph $G = (V, E)$ obtained from D by adding edges between every pair of non-adjacent vertices that have a common successor, and then dropping arc directions. The size of the edge set E is also reported in Table 3. After the application of pre-processing

					after pre-pro.	
network	origin	$\|V\|$	$\|A\|$	$\|E\|$	$\|V\|$	$\|E\|$
WILSON	Wilson's liver disease	21	23	27	-	-
ALARM	anesthesia monitoring	37	44	62	-	-
VSD	prognosis of ventricular septal defect in infants	38	51	61	-	-
OESOPHAGUS	staging of oesophageal cancer	42	57	68	-	-
OESOPHAGUS+	prediction of response to treatment of oesophageal cancer	67	117	194	26	121
MUNIN	interpretation of electromyographic findings	1003	1244	1662	175	471
ICEA	prediction of coverage by antibiotics of pathogens causing pneumonia	89	154	215	59	170
PATHFINDER	diagnosis of lymphatic disease	109	192	211	14	49

Table 3: Probabilistic network characteristics

techniques for computing the treewidth [8], an additional four instances to conduct our heuristics on are available. The size of these four instances is reported in Table 3 as well. After [8], henceforth we refer to these instances as INSTANCENAME_{3,4}.

Table 4 shows the results of the heuristics and lower bounds for these instances. The computation times for both lower bounds are neglectable. For three instances lower and upper bounds are equal and so the treewidth is reported. For the other instances the gap ranges from 1 to 5. Further the same observations can be made as for the frequency assignment instances. The computation time of `MSVS` is substantially larger than for the other heuristics, but at the same time the best result for all instances is obtained with this heuristic. Especially, for ICEA(_3) and MUNIN, `MSVS` outperforms the other heuristics.

Although not reported in Table 4, like for the frequency assignment instances, the range of

network	lower bounds		upper bounds				CPU time (in seconds)			
	MC	MMD	MCS	LEX_P	LEX_M	MSVS	MCS	LEX_P	LEX_M	MSVS
WILSON	2	2	3	3	3	3	0.00	0.00	0.01	0.01
ALARM	4	4	4	4	4	4	0.00	0.01	0.03	0.05
VSD	4	4	4	4	4	4	0.01	0.00	0.03	0.09
OESOPHAGUS	3	3	3	3	3	3	0.01	0.01	0.03	0.09
OESOPHAGUS+	9	9	10	11	10	10	0.02	0.04	0.20	1.28
MUNIN	3	3	10	16	16	8	24.99	30.69	286.46	5636.16
ICEA	5	5	15	14	13	9	0.10	0.10	0.55	4.62
PATHFINDER	5	5	7	7	7	6	0.04	0.05	0.28	0.36
OESOPHAGUS+_4	8	8	10	11	10	10	0.00	0.01	0.04	0.21
MUNIN_3	3	4	9	8	8	7	0.23	0.27	2.78	65.36
ICEA_3	5	5	15	14	13	9	0.05	0.06	0.25	2.86
PATHFINDER_4	5	5	7	7	7	6	0.00	0.00	0.01	0.03

Table 4: Computational results for Probabilistic Networks

bounds generated by the triangulation heuristics is smaller for MCS compared with LEX_P and LEX_M. This in particular holds for the MUNIN network. Where the bound ranges from 16 to 56 for LEX_P and LEX_M, MCS reports bounds from 10 to 26 (see [8] for details). Hence, it is important not to restrict the computations to a single run of these heuristics.

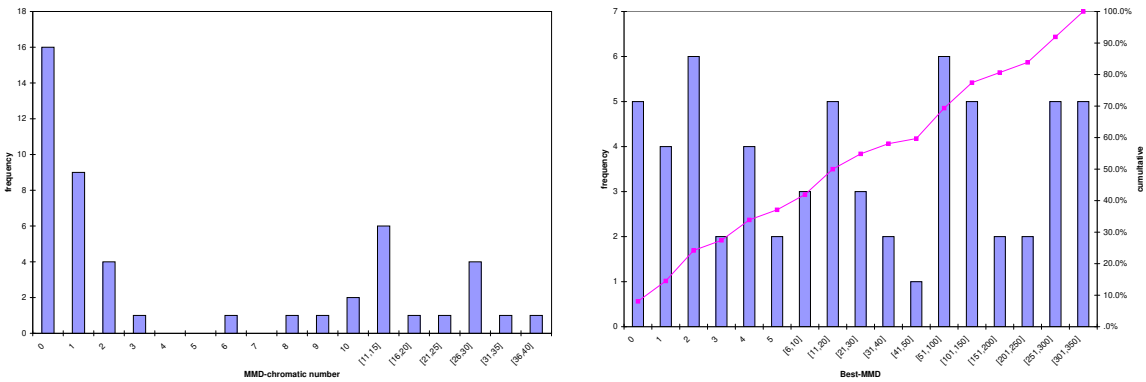
5.3 Vertex Coloring

Except for the frequency assignment instances and the probabilistic networks, where tree decomposition is indeed reported in the literature, we decided to test the algorithms also on another class of publicly available graphs. We have chosen for the well-known DIMACS benchmarks for vertex coloring [11][1]. Advantage of these instances is, that for many of them the chromatic number is known. In Section 4.2, we have seen that this value minus one is a lower bound for treewidth as well, and outperforms the clique bound. Details on the 62 instances and results can be found in Table 5 in Appendix A. In Figure 4 and 5, the results are reported in a different, more descriptive way. Figure 4 deals with the lower bounds and their relation to the upper bounds. Figure 5 depicts the quality of the upper bounds.

In Figure 4(a), we compare the chromatic number (provided the value is known) with the MMD bound. We counted the number of times the values are equal, differ one, differ two, etcetera. This results in the histogram of Figure 4(a). Note that the intervals are not equally sized. The diagram shows that in 33 of the 49 cases, the MMD bound is indeed better. The improvement of the lower bound is incidentally over 30.

In Figure 4(b), we compare the MMD bound with the best upper bound achieved by the various heuristics. Again, we count the number of times the difference between best upper bound and MMD lower bound is zero, one, etcetera. In addition, the cumulative percentage below a difference is displayed. Figure 4(b) shows that for 7 (out of 62) instances the optimal value is achieved, whereas for 25 instances (or 37%) the gap is at most 5. On the other side, the gap is extremely large (> 100) for another 22%, with a maximum gap of 327. For many instances it seems to be difficult to find a good lower bound, or a good upper bound, or both.

[1] The FLAT* instances as well as the larger DSJC* and DSJR* instances are left out, since they have a very high density and thus are very time consuming.



(a) MMD- chromatic number

(b) best upper bound - MMD

Figure 4: Statistical information lower bounds for DIMACS vertex coloring instances

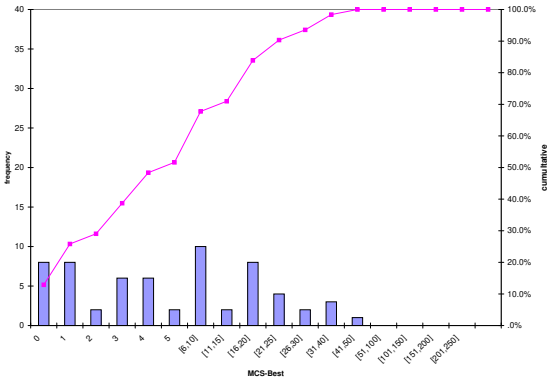
Figures 5(a)–5(d) show the difference between the best upper bound achieved and the heuristics MCS, LEX_P, LEX_M, and MSVS, respectively, in a similar way as the difference between best upper bound and MMD bound in Figure 4(b). To simplify a comparison between the histograms, the same intervals and the same scale are used in all diagrams. The number of times LEX_M and MSVS supply the best upper bound attracts our attention at first sight. Respectively, 36 and 33 times the best bound is reported by these heuristics. An even more impressive result is obtained by joining the results of LEX_M and MSVS. Together, they are responsible for 59 (out of 62) best upper bounds. The bounds reported by the other heuristics MCS and LEX_P are more spread out.

The maximum difference between best upper bound and the bound by a particular heuristic is smallest for MSVS with 25, followed by MCS with 47, LEX_M with 193, and LEX_P with 198. The maximum difference between best *lower* bound and the heuristics is for respectively MCS, LEX_P, LEX_M, and MSVS, 344, 330, 330, and 327.

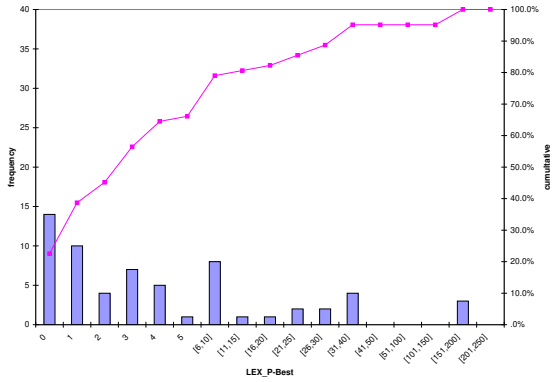
Concerning the computation times, again the MSVS heuristic is outperformed by all other heuristics. On average it took more than 20,000 seconds (> 5.5 hours) to compute the MSVS bound, whereas the MCS, LEX_P, and LEX_M bound are computed in respectively 1,768, 1,974, and 2,167 seconds (30–36 minutes). The computation of the MMD lower bound took on average a (in comparison with the other times) neglectable 1.17 seconds.

6 Concluding Remarks

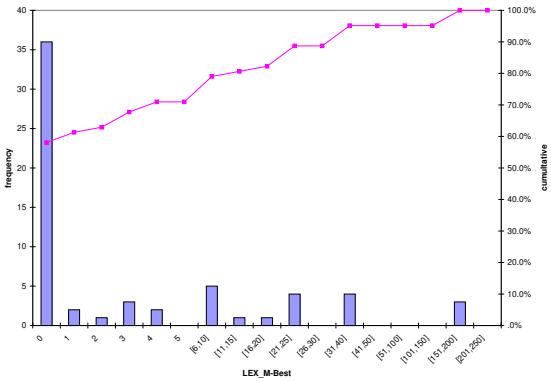
In recent years, alternatives for integer programming to solve combinatorial optimization problems attract more and more attention. One of these methods is based on a tree decomposition of the graph associated with the problem. In this approach, the treewidth of a graph has to be computed, independently of the application at hand. Studies to the solvability of treewidth have been rarely carried out. In this paper, a computational study to the quality of heuristics and lower bounds for treewidth has been undertaken. A total of four heuristics and two lower



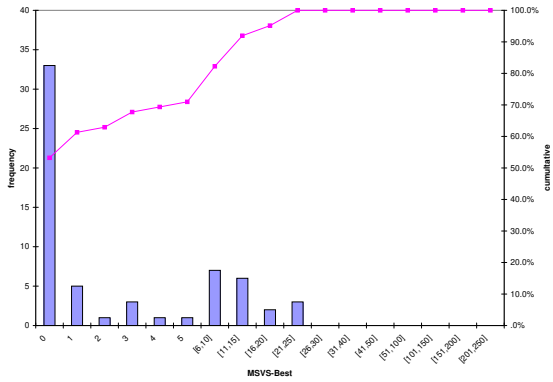
(a) MCS - best upper bound



(b) LEX_P - best upper bound



(c) LEX_M - best upper bound



(d) MSVS - best upper bound

Figure 5: Statistical information heuristics for DIMACS vertex coloring instances

bounds have been compared on graphs from different applications. Three of the heuristics are based on triangulation recognition algorithms, the other one on minimal separating vertex sets. The lower bounds compared are based on maximum cliques and the maximum minimum degree in a subgraph. For the latter one, it is easy to show that it is never worse than the maximum clique bound.

The bounds are tested on graphs from three different applications; frequency assignment, probabilistic networks, and vertex coloring. The best upper bounds were achieved by the MSVS heuristic. A good alternative to this time consuming bound is the LEX_M bound that performs almost as good as MSVS but has the advantage to be computable in substantially less time. The MMD algorithm provides a reasonable lower bound very fast.

For all applications, the treewidth of some graphs could be identified by combining the best lower and upper bound. On the other hand, for several graphs from frequency assignment and vertex

coloring the gap between best lower and upper bound is far too large to draw any conclusion about the treewidth. It is needless to say that for these instances better lower and upper bounds are necessary.

To determine the treewidth for graphs with a relatively small gap between lower and upper bound, also the development and implementation of exact solution procedures can be effective. Ideas in this direction are currently under development. Moreover, for all graphs pre-processing can be a powerful tool to simultaneously reduce the size of the graph and obtain lower bounds. In [8], a first attempt in this direction has been undertaken for the graphs from probabilistic networks (cf. Section 5.2). In a follow-up paper more ideas for graph reduction will be presented. Therefore, this paper should be viewed as one of the first (together with [8]) in a series of the authors and other collaborators to study the practical setting of treewidth and the tractability of the tree decomposition approach for combinatorial optimization problems.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, New Jersey, 1993.

[2] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability. *BIT*, 25:2–23, 1985.

[3] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[4] E. Balas and J. Xue. Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. *Algorithmica*, 15(5):397–412, 1996.

[5] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125(1–2):3–17, 2001.

[6] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1–2):1–21, 1993.

[7] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[8] H. L. Bodlaender, A. M. C. A. Koster, F. van den Eijkhof, and L. C. van der Gaag. Preprocessing for triangulation of probabilistic networks. In J. Breese and D. Koller, editors, *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 32–39, San Francisco, 2001. Morgan Kaufmann Publishers.

[9] EUCLID CALMA project. Publications available at FTP Site: `ftp.win.tue.nl`, Directory `/pub/techreports/CALMA`, 1995.

[10] W. Cook and P. D. Seymour. An algorithm for the ring-router problem. Technical report, Bellcore, 1994.

[11] The second DIMACS implementation challenge: $\mathcal{NP}$-Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability. See http://dimacs.rutgers.edu/Challenges/ or http://mat.gsia.cmu.edu/COLOR/instances.html, 1992–1993.

[12] G. A. Dirac. On rigid circuit graphs. *Abh. Math. Semin. Univ. Hamburg*, 25:71–76, 1961.

[13] A. Eisenblätter, M. Grötschel, and A. M. C. A. Koster. Frequency assignment and ramifications of coloring. *Discussiones Mathematicae Graph Theory*, to appear, 2001.

[14] A. Eisenblätter and A. M. C. A. Koster. FAP web - A website devoted to frequency assignment. URL: http://fap.zib.de, 2000.

[15] D. R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

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

[17] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, Inc., 1980.

[18] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of SIAM*, 9:551–570, 1961.

[19] I. V. Hicks. *Branch Decompositions and their Applications.* PhD thesis, Rice University, Houston, Texas, 2000.

[20] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local compuations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[21] A. M. C. A. Koster. *Frequency Assignment - Models and Algorithms.* PhD thesis, Maastricht University, 1999. Available at http://www.zib.de/koster/.

[22] A. M. C. A. Koster, C. P. M. van Hoesel, and A. W. J. Kolen. Solving frequency assignment problems via tree-decomposition. Technical Report RM 99/011, Maastricht University, 1999. Available at http://www.zib.de/koster/.

[23] A. M. C. A. Koster, C. P. M. van Hoesel, and A. W. J. Kolen. Lower bounds for minimum interference frequency assignment problems. *Ricerca Operativa*, 30(94–95):101–116, 2000.

[24] 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.

[25] D. R. Lick and A. T. White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22:1082–1096, 1970.

[26] K. Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927.

[27] P. M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, 1994.

[28] N. Robertson and P. D. Seymour. Graph minors. I. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35:39–61, 1983.

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

[30] N. Robertson and P. D. Seymour. Graph minors. X. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52:153–190, 1991.

[31] N. Robertson and P. D. Seymour. Graph minors. XIII. the disjoint paths problems. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

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

[33] 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.

[34] G. Szekeres and H. S. Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4:1–3, 1968.

[35] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[36] P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proc. 16th Ann. Symp. on Foundations of Computer Science*, pages 75–84, 1975.

[37] B. Verweij. *Selected Applications of Integer Programming: A Computational Study.* PhD thesis, Utrecht University, Utrecht, The Netherlands, 2000.

A Results for vertex coloring instances

instance	$\|V\|$	$\|E\|$	lower bounds		upper bounds				CPU time (in seconds)				
			$\chi(G)-1$	MMD	MCS	LEX_P	LEX_M	MSVS	MMD	MCS	LEX_P	LEX_M	MSVS
ANNA	138	986	10	10	13	13	12	12	0.01	0.14	0.16	1.24	18.39
DAVID	87	812	10	10	14	13	13	13	0.00	0.08	0.10	0.56	7.77
HUCK	74	602	10	10	10	10	10	10	0.00	0.04	0.05	0.24	2.30
HOMER	561	3258	12	12	38	37	37	31	0.04	10.09	14.24	68.08	556.82
JEAN	80	508	9	9	9	9	9	9	0.00	0.04	0.04	0.29	1.98
GAMES120	120	638	8	8	37	37	37	51	0.00	2.16	2.61	5.20	65.97
QUEEN5_5	25	160	4	12	19	18	18	18	0.00	0.01	0.02	0.04	0.22
QUEEN6_6	36	290	6	15	27	27	26	28	0.00	0.07	0.07	0.16	1.16
QUEEN7_7	49	476	6	18	38	36	35	38	0.01	0.27	0.23	0.51	4.66
QUEEN8_8	64	728	8	21	51	48	46	49	0.00	0.94	0.81	1.49	16.38
QUEEN9_9	81	1056	9	24	66	60	59	66	0.01	2.82	2.41	3.91	47.35
QUEEN10_10	100	1470	-	27	82	76	73	79	0.01	7.87	6.61	9.97	128.30
QUEEN11_11	121	1980	10	30	96	92	89	101	0.01	19.47	16.45	23.36	310.83
QUEEN12_12	144	2596	-	33	122	109	106	120	0.02	46.11	38.77	49.93	702.29
QUEEN13_13	169	3328	12	36	139	129	125	145	0.04	99.90	85.01	107.62	1589.77
QUEEN14_14	196	4186	-	39	163	149	145	164	0.07	211.08	177.79	215.36	3275.75
QUEEN15_15	225	5180	-	42	183	173	167	192	0.11	416.44	347.58	416.25	6002.33
QUEEN16_16	256	6320	-	45	210	195	191	214	0.21	796.98	657.53	773.09	11783.30
FPSOL2.I.1	269	11654	64	64	66	66	66	66	1.03	50.18	194.66	319.34	4220.91
FPSOL2.I.2	363	8691	29	31	35	57	52	31	0.51	21.75	541.81	622.22	8068.88
FPSOL2.I.3	363	8688	29	31	35	57	52	31	0.67	23.13	555.27	621.89	8161.78
INITHX.I.1	519	18707	53	55	56	223	223	56	5.83	109.75	3133.94	3144.95	37455.10
INITHX.I.2	558	13979	30	31	42	233	228	35	2.37	54.56	4715.02	5567.96	37437.20
INITHX.I.3	559	13969	30	31	42	233	228	35	2.48	56.22	4757.95	5190.39	36566.80
MILES1000	128	3216	41	41	55	51	49	53	0.07	5.48	5.15	14.39	229.00
MILES1500	128	5198	72	72	80	77	77	83	0.10	10.63	11.23	29.12	268.19
MILES250	125	387	7	7	10	10	10	9	0.00	0.11	0.12	1.12	10.62
MILES500	128	1170	19	19	26	23	22	28	0.01	0.83	0.84	4.37	87.18
MILES750	128	2113	30	31	41	37	37	38	0.02	2.64	2.42	8.13	136.69
MULSOL.I.1	138	3925	48	48	51	66	66	50	0.07	4.22	8.99	17.77	240.24
MULSOL.I.2	173	3885	30	31	35	69	69	32	0.09	4.24	18.92	34.06	508.71
MULSOL.I.3	174	3916	30	31	35	69	69	32	0.09	4.32	19.40	34.58	527.89
MULSOL.I.4	175	3946	30	31	35	69	69	32	0.09	4.42	19.90	35.53	535.72
MULSOL.I.5	176	3973	30	31	35	69	69	31	0.09	4.47	20.41	36.25	549.55
MYCIEL3	11	20	3	3	5	5	5	5	0.00	0.00	0.00	0.00	0.01
MYCIEL4	23	71	4	5	11	10	11	11	0.00	0.01	0.01	0.02	0.13
MYCIEL5	47	236	5	8	21	23	23	20	0.00	0.06	0.11	0.28	2.00
MYCIEL6	95	755	6	12	38	46	47	35	0.01	0.94	2.59	4.56	29.83
MYCIEL7	191	2360	7	18	69	93	94	74	0.03	16.98	72.44	109.86	634.32
SCHOOL1	385	19095	-	73	273	254	252	244	4.79	2770.65	3299.87	3987.64	41141.10
SCHOOL1_NSH	352	14612	-	61	225	193	192	214	3.08	1653.17	1700.36	2059.52	28954.90
ZEROIN.I.1	126	4100	48	48	50	50	50	50	0.17	5.33	8.96	17.78	338.26
ZEROIN.I.2	157	3541	29	29	42	40	40	33	0.09	4.27	16.12	24.82	448.74
ZEROIN.I.3	157	3540	29	29	42	40	40	33	0.07	4.19	16.03	24.69	437.06
LE450_5A	450	5714	4	17	330	308	310	317	0.35	8340.05	7446.27	7836.99	73239.66
LE450_5B	450	5734	4	17	331	314	313	320	0.46	8593.04	7499.38	7909.11	73644.28
LE450_5C	450	9803	4	33	369	345	348	340	1.38	10819.96	9953.81	10745.70	103637.17
LE450_5D	450	9757	4	32	373	348	349	326	2.30	11141.73	9894.74	10681.29	96227.40
LE450_15A	450	8168	14	24	327	299	296	297	0.39	6350.69	6461.77	6887.15	59277.90
LE450_15B	450	8169	14	24	317	300	296	307	0.80	6067.16	6484.28	6886.84	65173.20
LE450_15C	450	16680	14	49	393	379	379	376	7.58	10974	11324.23	12471.09	122069.00
LE450_15D	450	16750	14	51	392	377	379	375	8.76	10804	11313.32	12481.22	127602.00
LE450_25A	450	8260	24	26	275	255	255	270	1.31	3750.33	4194.31	4478.30	53076.40
LE450_25B	450	8263	24	25	288	257	251	264	1.20	4913.72	4636.22	4869.97	52890.00
LE450_25C	450	17343	24	52	377	359	355	365	8.13	9182.24	9992.39	10998.68	109141.00
LE450_25D	450	17425	24	51	377	359	356	359	5.31	10227.34	10658.33	11376.02	111432.25
DSJC125.1	125	736	-	8	73	73	70	67	0.01	9.98	9.03	12.90	171.54
DSJC125.5	125	3891	-	53	114	111	110	110	0.07	24.69	23.90	38.07	254.90
DSJC125.9	125	6961	-	103	121	119	119	120	0.20	26.65	26.93	55.60	70.79
DSJC250.1	250	3218	-	18	193	186	183	179	0.09	477.40	462.75	528.10	5507.86
DSJC250.5	250	15668	-	109	238	235	233	233	3.62	749.19	749.09	1111.66	7756.38
DSJC250.9	250	27897	-	211	245	244	243	244	8.20	789.43	788.35	1414.58	1684.83

Table 5: Computational results for DIMACS vertex coloring instances