Treewidth Computations II. Lower Bounds

Hans L. Bodlaender

Arie M. C. A. Koster

Technical Report UU-CS-2010-022
September 2010

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 II.
Lower Bounds

Hans L. Bodlaender[*] Arie M. C. A. Koster[†]

Abstract

For several 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.

Good lower bounds on the treewidth of a graph can, amongst others, help to speed up branch and bound algorithms that compute the treewidth of a graph exactly. A high lower bound for a specific graph instance can tell that a dynamic programming approach for solving a problem is infeasible for this instance.

This paper gives an overview of several recent methods that give lower bounds on the treewidth of graphs.

1 Introduction

For many applications, there is the wish to be able to compute the treewidth of a given graph efficiently, and to find a tree decomposition of the graph with optimal or close to optimal width. While some theoretical results appear to be not useful in a practical setting (e.g., see the experimental evaluation by Röhrig [31] of the linear time fixed parameter algorithm of [5]), there are several algorithms that are fast, not hard to implement, and give reasonably good results for graphs derived from real life applications. In this short series of papers (see [12, 13]), we discuss such practical algorithms for computing the treewidth.

In this second paper in the series, we focus on algorithms that compute *lower bounds* on the treewidth of the input graphs. There are several reasons, why it is desirable to have good lower bounds on the treewidth of the input graph:

- A lower bound heuristic can be used as a subroutine in a branch and bound algorithm: high lower bounds can tell that certain branches can be cut of in the search. See for

[*]Department of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. hansb@cs.uu.nl

[†]Lehrstuhl II für Mathematik, RWTH Aachen University, Wüllnerstr. 5b, D-52062 Aachen, Germany. koster@math2.rwth-aachen.de

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

instance the work of Gogate and Dechter [20] or the recent work of Cole Smith et al. [18] on branchwidth.

- Suppose we want to solve a combinatorial problem on a graph G. Further suppose it is known that for fixed k, there is an efficient, e.g., linear time dynamic programming algorithm that solves this problem on graphs given with a tree decomposition of width at most k. If we have a high lower bound on the treewidth of G, then this tells us that using this dynamic programming algorithm will use much time, and hence is not a feasible approach.
- Lower bounds give indications of the quality of treewidth upper bounds and upper bound algorithms.
- Better lower bounds can make preprocessing more effective. There are preprocessing rules that apply only when a large enough lower bound on the treewidth of the graph is known.

The first paper in this series [12] discusses upper bound heuristics: algorithms that yield tree decompositions whose width is not necessarily optimal, but often comes close to the optimal width. A third paper that focuses on exact algorithms and preprocessing methods is planned [13].

This paper is structured as follows. In Section 2, we give a few necessary definitions. In Section 3, we give a number of elementary lower bound methods for the treewidth, based on the degree of vertices, and one bound based on both the average degree and the girth. While the bounds obtained by these methods usually are rather small, they can often be improved considerably by considering the bounds over induced subgraphs (as discussed in Section 4) or minors / contractions (see Section 5) of the input graph. Slower but usually better than the elementary bounds is a lower bound based on Maximum Cardinality Search (Section 7). This bound can also be combined with taking induced subgraphs and minors/contractions. A technique that can be used in combination of each of these methods is one, introduced by Clautiaux et al. [17]; in this technique we use another lower bound algorithm on a graph obtained by adding certain edges to the input graph. In Section 8, the technique is explained, and a slower but more powerful variant that combines the technique with another level of taking contractions is discussed. Two rather different lower bound techniques are discussed in Section 9 and 6. In Section 9, a different characterisation of the notion of treewidth in terms of *brambles* is used to obtain treewidth lower bounds. Section 6 discusses a lower bound based on the second smallest eigenvalue of the Laplacian matrix of the graph. A performance analysis of a selection of the lower bounds is reported in Section 11. The paper is completed with some miscellaneous ideas for lower bound methods (Section 10) and some conclusions (Section 12).

2 Preliminaries

Throughout this graph, $G = (V, E)$ denotes a simple undirected graph. $n = |V|$ denotes the number of vertices of G, and $m = |E|$ denotes the number of edges of G. We assume that the reader is familiar with standard graph notions like degree, path, cycle, clique, planarity. For a graph $G = (V, E)$ and vertex set $W \subseteq V$, the subgraph of G induced by W is denoted $G[W] = (W, \{\{v, w\} \in E \mid v, w \in W\})$.

The degree of a vertex $v \in V$ in a graph $G = (V, E)$ is denoted by $d_G(v)$, or, when G is clear from the context, by $d(v)$.

The operation of *contracting an edge* $\{v, w\} \in E$ in a graph $G = (V, E)$ builds the graph, obtained from G by taking a new vertex x that is adjacent to all neighbours of v and of w, and removing v, w, and its incident edges. A graph H is said to be a *contraction* of G, if H can be obtained from G by applying zero or more contractions of edges. (See [41, 42] for more details on the notion of contraction.) H is a *minor* of G if H is the contraction of a subgraph of G, i.e., H can be obtained from G by zero or more vertex deletions, edge deletions, and/or edge contractions.

We say that a collection of paths from v to w in a graph $G = (V, E)$ are *vertex disjoint*, if there is no vertex in $V - \{v, w\}$ that belongs to more than one of the paths.

The notions of treewidth and tree decomposition were introduced by Robertson and Seymour [30] in their work on graph minors. Similar notions were invented earlier, e.g., the notion of partial k-tree, see e.g., [2]. For an overview of related notions, see [6].

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 $X_{i_0} \cap X_{i_2} \subseteq X_{i_1}$.

The following simple lemma shows that we always can find a tree decomposition that is in a certain sense 'minimal'.

Lemma 2 *Let $G = (V, E)$ be a graph of treewidth at most k. Then G has a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of width at most k such that for all $(i, j) \in F$: $X_i \not\subseteq X_j$ and $X_j \not\subseteq X_i$.*

Proof: When we have a tree decomposition of G with two adjacent bags i, j with $X_i \subseteq X_j$, then we can contract the edge (i, j) in T: replace the bags for i and j by one bag containing X_j and make this adjacent to all neighbours of i and of j. This gives again a tree decomposition of G with the same width.

Repeating the operation gives the required tree decomposition. □

The following well known fact shows that we can restrict ourselves to algorithms on connected graphs. For a proof, see e.g., [6].

Lemma 3 *The treewidth of a graph equals the maximum treewidth over its connected components.*

3 Degree Based Bounds

In this section, we give three elementary lower bounds, based upon the degrees of vertices in the graph, and a fourth one of which we omit the proof. The first two are based upon folklore results on treewidth; the third is due to Ramachandramurthi [27, 28]. A fourth bound by Sunil Chandran and Subramanian [38] uses both the average degree of vertices and the girth. We give the proof for the first three bounds.

3.1 Smallest Degrees

Definition 4 *(i). The minimum degree of a vertex in $G = (V, E)$ is denoted $\delta(G) = \min_{v \in V} d(v)$.*

(ii). The second smallest degree of a vertex in $G = (V, E)$ is denoted $\delta_2(G) = \min\{d(v) \mid v \in V \wedge \exists w \in V : d(w) \leq d(v)\}$.

The third lower bound, due to Ramachandramurthi [27,28] is, when $G = (V, E)$ is a clique, the number of vertices minus 1; and when G is not a clique, the minimum over all pairs of non-adjacent vertices v, w, of the maximum of the degree of v and w.

Definition 5 *Let $G = (V, E)$ be a graph. We define $\gamma_R(G)$ to be the value*

$$\gamma_R(G) = \begin{cases} |V| - 1 & \textit{if } G \textit{ is a clique} \\ \min_{v,w \in V, \{v,w\} \notin E} \max\{d(v), d(w)\} & \textit{otherwise} \end{cases}$$

Lemma 6 *(i). Let $G = (V, E)$ be a graph of treewidth at most k. Then G has a vertex of degree at most k.*

(ii). Let $G = (V, E)$ be a graph of treewidth at most k that contains at least two vertices. Then G contains at least two vertices of degree at most k.

(iii). Let $G = (V, E)$ be a graph of treewidth at most k that is not a clique. Then G contains at least two vertices v, w of degree at most k such that v and w are not adjacent.

Proof: Take a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ as implied by Lemma 2. If there is only one bag, the lemma clearly holds, as there are at most $k+1$ vertices in G, so we can take respectively an arbitrary, two arbitrary, and two arbitrary non-adjacent vertices.

Suppose T has at least two bags. Take a bag i that is a leaf of T, and let j be its neighbour. Take a vertex $v \in X_i - X_j$ — it is implied by Lemma 2 that such a v exists. v cannot belong to any other bag except X_i, otherwise it would belong to X_j by the definition of tree decomposition. So, all neighbours of v must belong to X_i, and hence v has degree at most $|X_i| - 1 \leq k$. This proves (i).

T has at least two leaves. Take a leaf $i' \neq i$, with neighbour j' in T, and take $v' \in X_{i'} - X_{j'}$. As above, v' only belongs to $X_{i'}$ and has degree at most k. As there is no bag that contains both v and w, $\{v, w\}$ is not an edge in G; this proves (ii) and (iii). $\square$

Parts (i) and (ii) of Lemma 6 are not necessary for the proof of the next lemma. They can be used to give direct proofs that $\delta(G)$ and $\delta_2(G)$ are lower bounds for the treewidth.

Lemma 7 (Ramachandramurthi [26–28]) *For each graph $G = (V, E)$, the treewidth of G is at least $\gamma_R(G) \geq \delta_2(G) \geq \delta(G)$.*

Proof: If G is a clique, then the treewidth of G equals $|V| - 1 = \gamma_R(G)$. Suppose G is not a clique. Let the treewidth of G be k. By Lemma 6, there are two non-adjacent vertices v, $w \in V$ with degree at most k; hence $k \leq \gamma_R(G)$.

$\gamma_R(G) \geq \delta_2(G) \geq \delta(G)$ follows easily from the definitions. $\square$

It is trivial that computing $\delta(G)$ and $\delta_2(G)$ can be done in linear time for a given graph G. It is also possible to compute $\gamma_R(G)$ in linear time; two different algorithms can be found in [26] and in [23, 41].

3.2 Average Degree and Girth

Sunil Chandran and Subramanian [38] found a non-trivial lower bound on the treewidth that is based both on the average degree of the vertices, and on the girth. The *girth* of a graph $G = (V, E)$ is the minimum length of a cycle in G.

Theorem 8 (Sunil Chandran and Subramanian [38]) *Let $G = (V, E)$ be a graph. Suppose the girth of G is g, and the average degree of vertices in G is δ. Then the treewidth of G is at least*

$$\max\left(\left\lceil \frac{\delta}{2} \right\rceil, \frac{1}{4e(g+1)}(\delta - 1)^{\lfloor (g-1)/2 \rfloor} - 2\right).$$

Note that in the formula above, e denotes the base of the natural logarithm, $2.71828\ldots$. As the degeneracy of a graph is at least half its average degree, in particular, the second term is interesting. The girth of a graph can be computed in polynomial time using shortest paths techniques. Most of the graphs that were used in our experiments contain a cycle of length three, i.e., have girth 3. In such a case, the bound given by Theorem 8 is $\max(\lceil \frac{\delta}{2} \rceil, \frac{d-1}{16e} - 2) = \lceil \frac{\delta}{2} \rceil$, which is smaller than the degeneracy of G, i.e., in all the cases that a graph has girth 3, the bound of Theorem 8 is inferior to the degeneracy.

4 Taking Subgraphs

4.1 Degree Based Bounds and Subgraphs

One or two small degree vertices in a graph G can already cause that the value of $\delta(G)$, $\delta_2(G)$, or $\gamma_R(G)$ is small. For getting a larger lower bound, it possibly would have been better if there were not such vertices at all. Thus, what we can do is deleting vertices with small degree, and looking to lower bounds on thus obtained induced subgraphs. Simple but essential is the following well-known observation.

Lemma 9 *Let $G = (V, E)$ be a graph, and $W \subseteq V$ be a set of vertices. The treewidth of $G[W]$ is at most the treewidth of G.*

Proof: Suppose $(\{X_i \mid i \in I\}, T = (I, F))$ is a tree decomposition of G of width k. Now, one easily verifies that $(\{Y_i \mid i \in I\}, T = (I, F))$ with for all $i \in I$, $Y_i = X_i \cap W$ is a tree decomposition of $G[W]$ of width at most k. □

Corollary 10 *Let H be a subgraph of G. Then the treewidth of H is at most the treewidth of G.*

Proof: Note that if $E' \subseteq E$, then each tree decomposition of a graph $G = (V, E)$ is also a tree decomposition of $G' = (V, E')$. Now use Lemma 9. □

Combining the bounds $\delta(G)$, $\delta_2(G)$, and $\gamma_R(G)$ with taking subgraphs gives three new treewidth lower bounds.

Definition 11 *(i). The* degeneracy *of a graph $G = (V, E)$, denoted $\delta D(G)$, is the maximum of $\delta(H)$ over all subgraphs H of G.*

(ii). The δ_2-degeneracy *of a graph $G = (V, E)$ with $|V| \geq 2$, denoted $\delta_2 D(G)$, is the maximum of $\delta_2(H)$ over all subgraphs H of G with at least two vertices*

(iii). The γ_R-degeneracy *of a graph $G = (V, E)$ with $|V| \geq 2$ denoted $\delta_2 D(G)$, is the maximum of $\gamma_R(H)$ over all subgraphs H of G with at least two vertices.*

If we combine Lemma 7 with Corollary 10, we obtain directly the following result.

Lemma 12 *For each graph $G = (V, E)$, the treewidth of G is at least $\gamma_R D(G) \geq \delta_2 D(G) \geq \delta D(G)$.*

There is an easy and well known method to compute the degeneracy of a graph: continuously remove a vertex of smallest degree of the graph, and return the largest degree thus seen. See Algorithm 1. We term the algorithm MMD (Maximum Minimum Degree).

The following result is ‘folklore’.

Lemma 13 *MMD(G) as in Algorithm 1 computes the degeneracy of G.*

Algorithm 1 MMD(Graph G)

```
$H = G$
maxmin = 0
**while** $H$ has at least two vertices **do**
  Select a vertex $v$ from $H$ that has minimum degree in $H$.
  maxmin = max( maxmin, $d_H(v)$ ).
  Remove $v$ and its incident edges from $H$.
**end while**
**return** maxmin
```

Proof: The output of the algorithm is at most the degeneracy: when maxmin is set to a value k, then H, which is a subgraph of G, has only vertices of degree at least k. The output is at least the degeneracy: suppose $G' = (V', E')$ is a subgraph of G with minimum degree $\delta D(G)$. Suppose $v \in V'$ is the first vertex in V' that is removed. Then, at the moment v' is removed, all vertices in H have a degree which is at least $d_H(v') \geq d_{G'}(v') \geq \delta D(G)$, so maxmin will be at least $\delta D(G)$ after this step. □

It is possible to compute $\delta_2 D(G)$ for a given graph $G = (V, E)$ in $O(nm)$ time, as was shown by Koster, Wolle, and Bodlaender in [23], see also [41]. The idea of the algorithm is that for each vertex $v \in V$, we look at the case where we assume will be the minimum degree vertex in a subgraph H of G with $\delta_2(H) = \delta_2 D(G)$. We now repeatedly remove from G a vertex w whose degree is minimum among all vertices except v, and return the largest degree of a vertex at its time of removal. The largest bound over all choices of v indeed equals $\delta_2 D(G)$.

Algorithm 2 Delta2D(Graph G)

```
d2D = 0
**for** all vertices $v \in V$ **do**
  $H = G$
  **while** $H$ has at least two vertices **do**
    Select a vertex $w \neq v$ from $H$ that has minimum degree in $H$ amongst all vertex
    except $v$.
    maxmin = max(maxmin, $d_H(v)$).
    Remove $v$ and its incident edges from $H$.
  **end while**
**end for**
**return** maxmin
```

Lemma 14 (Koster, Wolle, and Bodlaender [23, 41]) *Delta2D(G) from Algorithm 2 computes $\delta_2 D(G)$ and has an implementation that uses $O(nm)$ time.*

In [23], it has been shown that it is NP-complete to decide, for given graph $G = (V, E)$ and given k, if the γ_R-degeneracy of G is at least k. Several relations between the parameters and related results can also be found in [23, 41].

4.2 Subgraphs with Approximation or Exact Algorithms

If a graph G is planar, the branchwidth of G can be computed exactly in polynomial time [34], see also [21]. As treewidth and branchwidth differ at most a multiplicative factor of $\frac{3}{2}$, this gives an approximation of and hence also a lower bound for the treewidth of G. If G is not planar, but 'close to planar', we could first find a maximal planar subgraph H of G, and then compute the branchwidth and a lower bound for the treewidth of H. This is also a lower bound for the treewidth of G. For an overview paper on branchwidth, see [22].

Instead of looking for a maximal planar subgraph, we can also look for a subgraph H of G with another property that allows to compute the treewidth exactly or with a good approximation algorithm; such an approach probably is only useful in specific contexts (e.g., if we know that G is formed by adding a few edges to a permutation graph — we can compute the treewidth exactly for permutation graphs [10]).

5 Taking Minors

In the previous section, we looked at the improvement of lower bounds by taking subgraphs. However, much stronger lower bounds often can be obtained by using contractions. The following result is well known.

Lemma 15 *Let $H = (W, F)$ be a minor of $G = (V, E)$. Then the treewidth of H is at most the treewidth of G.*

Proof: As shown in Lemma 9 and Corollary 10, deleting vertices and edges cannot increase the treewidth. Contracting an edge also cannot increase the treewidth: Suppose $(\{X_i \mid i \in I\}, T = (I, F))$ is a tree decomposition of $G = (V, E)$ of width k. Suppose we contract edge $\{v, w\}$ to vertex x. We can build a tree decomposition of the new graph by replacing each occurrence of v or w in a bag by x, i.e., set $X'_i = X_i$ if $X_i \cap \{v, w\} = \emptyset$, and $X'_i = X_i - \{v, w\} \cup \{x\}$ if $X_i \cap \{v, w\} \neq \emptyset$. The width of the tree decomposition $(\{X'_i \mid i \in I\}, T = (I, F))$ is at most k. The lemma now follows. $\Box$

Thus, given a treewidth lower bound, we can build a new treewidth lower bound by looking at the maximum of this bound over all minors.

Definition 16 *(i). The* contraction degeneracy *of a graph $G = (V, E)$, denoted $\delta C(G)$, is the maximum of $\delta(H)$ over all minors H of G.*

(ii). The δ_2-contraction degeneracy *of a graph $G = (V, E)$ with $|V| \geq 2$, denoted $\delta_2 C(G)$, is the maximum of $\delta_2(H)$ over all minors H of G with at least two vertices*

*(iii). The γ_R-*contraction degeneracy *of a graph $G = (V, E)$ with $|V| \geq 2$ denoted $\delta_2C(G)$, is the maximum of $\gamma_R(H)$ over all minors H of G with at least two vertices.*

Combining Lemma 7 and Lemma 15 directly gives the following result.

Lemma 17 *For each graph $G = (V, E)$, the treewidth of G is at least $\gamma_RC(G) \geq \delta_2C(G) \geq \delta C(G)$.*

Unfortunately, each of the parameters δC, δ_2C, and γ_RC is NP-hard to compute [16, 23]. We can however use heuristics for these parameters. A simple heuristic for computing the contraction degeneracy can be seen as a variant of the MMD algorithm, and is hence termed MMD+. See Algorithm 3.

Algorithm 3 MMD+(Graph G)

```
  $H = G$
  maxmin = 0
  while $H$ has at least two vertices do
    Select a vertex $v$ from $H$ that has minimum degree in $H$.
    maxmin = max( maxmin, $d_H(v)$ ).
    Select a neighbour $w$ of $v$. {A specific strategy can be used here.}
    Contract the edge $\{v, w\}$ in $H$.
  end while
  return maxmin
```

Note that the difference between the MMD and the MMD+ algorithm is that MMD deletes a vertex, while MMD+ contracts the vertex to a neighbour.

Lemma 18 *The MMD+ algorithm of Algorithm 3 outputs a value that is a lower bound on the contraction degeneracy of the input graph G, and hence a lower bound on the treewidth of G.*

Proof: The lemma follows directly by observing that at each step we have that H is a contraction (hence a minor) of G. □

The MMD+ treewidth lower bound was proposed independently by Bodlaender, Koster, and Wolle in [16], and by Gogate and Dechter in [20]. Bodlaender, Koster, and Wolle [16] consider three different heuristic strategies to select the neighbour w to which the minimum degree vertex v is contracted.

- The *min-d* strategy selects a neighbour of minimum degree. The strategy is motivated by the wish to increase the degree of small degree vertices as fast as possible.
- The *max-d* strategy selects a neighbour of maximum degree. In this way, we create some neighbours of very large degree.

- The *least-c* strategy selects the neighbour w of v such that v and w have a minimum of common neighbours. For each common neighbour x of v and w, contracting $\{v, w\}$ let the two edges $\{v, x\}$ and $\{w, x\}$ become the same edge. The strategy hence tries to contract an edge such that as few as possible edges disappear from the graph.

The resulting heuristics are called MMD+(min-d), MMD+(max-d), and MMD+(least-c). An experimental study of these reveals that the max-d strategy is inferior to the other two, and that the least-c strategy is usually the best of these.

In [16,41], for each of these heuristics, examples are given where the heuristics perform bad: the contraction degeneracy and treewidth of the graphs in the examples are of size $\Theta(\sqrt{n})$, while the heuristics can give a lower bound value of three. See [15,16,23,41,43] for more results on the contraction degeneracy and the related notions.

Contraction with Exact Algorithms Gives Lower Bounds In [8], it is observed that taking minors can be combined with an exact algorithm for treewidth to obtain good lower bounds. Suppose we have an algorithm A, that computes the treewidth of a given graph exactly, in worst case exponential time. Suppose also that we have a graph G, for which running A on G is infeasible (as the algorithm uses too much time or memory). Now, we obtain a minor H of G, by contracting edges of G, e.g., with the least-c strategy, till H is small enough to make it feasible to run A on H. Thus, we compute the exact treewidth of H, and obtain a lower bound on the treewidth of G. In [8], this approach is used with an exact dynamic programming algorithm for treewidth, but of course, the approach can also be used when we have another exact treewidth algorithm, e.g., one that uses branch and bound.

6 A Spectral Lower Bound

Sunil Chandran and Subramanian [37] have given a lower bound on the treewidth based on the second smallest eigenvalue of the Laplacian matrix of the graph.

Suppose we have a graph $G = (V, E)$. Number the vertices of G: $v_1, v_2, \ldots, v_n$. The adjacency matrix A of G is the n by n matrix with $A_{ij} = 1$ if $\{v_i, v_j\} \in E$, and $A_{ij} = 0$ if $\{v_i, v_j\} \notin E$. Take the n by n diagonal matrix D with D_{ii} the degree of v_i in G for all i, $1 \leq i \leq n$; $D_{ij} = 0$ whenever $i \neq j$. The *Laplacian matrix* of G is the matrix $L = D - A$. I.e., for all $i, j \in \{1, \ldots, n\}$, we have

$$L_{ij} = \begin{cases} d(v_i) & \text{if } i = j \\ 0 & \text{if } i \neq j \text{ and } \{v_i, v_j\} \notin E \\ -1 & \text{if } i \neq j \text{ and } \{v_i, v_j\} \in E \end{cases}$$

Theorem 19 (Sunil Chandran and Subramanian [37]) *Let $G = (V, E)$ and let μ be the second smallest eigenvalue of the Laplacian matrix of G. Let Δ be the maximum degree of a vertex in G. Then the treewidth of G is at least*

$$\left\lfloor \frac{3n}{4}\left(\frac{\mu}{\Delta + 2\mu}\right) \right\rfloor - 1$$

As μ can be computed in polynomial time, the result gives a polynomial time computable lower bound for treewidth. In the experiments that we conducted, the second smallest eigenvalue of the Laplacian matrix was in most cases in the interval $[0, 1]$. In such a case, the bound provided by Theorem 19 is at most 2, and thus of very little use. In other cases, the lower bound can be of good use, e.g., Sunil Chandran and Subramanian [37] use their result to show good lower bounds for the treewidth of hypercubes and Hamming graphs.

7 Maximum Cardinality Search

The Maximum Cardinality Search algorithm was introduced by Tarjan and Yannakakis [39] for recognising chordal (or triangulated) graphs. It is also sometimes used as an upper bound heuristic for treewidth; see e.g., the discussion of it in [12]. Surprisingly, it can also be used as a lower bound mechanism for treewidth, as was shown by Lucena [25].

Maximum Cardinality Search is a method to visit all vertices of a graph in some order. The algorithm works as follows. Initially, no vertex of the graph is visited. We now repeatedly visit an unvisited vertex, until all vertices are visited, following the maximum cardinality rule: each step, we must choose one of the unvisited vertices that has the largest number of visited neighbours. Note that a graph (with more than one vertex) has more than one ordering that can be generated by the Maximum Cardinality Search algorithm: we can start at any vertex, and when there are more unvisited vertices that are adjacent to the maximum number of visited vertices, the algorithm may choose any of these. The Maximum Cardinality Search algorithm thus generates an ordering of the vertices of the graph (bijection $V :\rightarrow \{1, \cdots, |V|\}$). Let an *MCS-ordering* of $G = (V, E)$ be an ordering of the vertices that can be generated by the MCS algorithm.

Lucena has shown that any MCS-ordering yields us a lower bound on the treewidth of G. For the proof of the following result, we refer to [24, 25].

Theorem 20 (Lucena [24, 25]) *Suppose the Maximum Cardinality Search algorithm on graph $G = (V, E)$ visits a vertex $v \in V$ that is, at the time of its visit, adjacent to k visited vertices. Then the treewidth of G is at least k.*

The theorem can be restated as follows:

Theorem 21 (Lucena [24, 25]) *For any graph $G = (V, E)$ and MCS-ordering π of G, the treewidth of G is at least*

$$MCSLB(G, \pi) = \max_{v \in V} |\{\{w, v\} \in E \mid \pi(w) < \pi(v)\}|$$

Note that $MCSLB(G, \pi)$ is the maximum over all vertices $v \in V$ of the number of visited neighbours of v at the time v is visited. As treewidth lower bound heuristic, we can thus generate an MCS-ordering π and compute the value $MCSLB(G, \pi)$. In [11], different tie-breaking rules were tried out for selecting the next vertex in the MCS-ordering (when more vertices have the same number of visited neighbours), but the results appear to be inconclusive.

Unfortunately, it is NP-hard to select from all MCS-orderings one that gives the largest MCSLB treewidth lower bound.

Theorem 22 (Bodlaender and Koster [11]) *For each fixed $k \geq 7$, it is NP-complete to decide for a given graph $G = (V, E)$, whether there exists an MCS-ordering π of G with $MCSLB(G, \pi) \geq k$.*

Combining MCSLB with contraction Similar to the degree based lower bounds from Section 3, we can combine the lower bound based on Maximum Cardinality Search with taking subgraphs, or (preferably) with contractions. The MCSLB+ heuristic works as follows: while G has at least one edge, we compute an MCS-ordering of G, and then contract an edge in G, and repeat. The largest bound found is the yielded treewidth lower bound. The combination of MCSLB with contraction was first proposed by Lucena [24]; experiments were reported in [24] and [16].

Finding a contraction or minor H of G and an MCS-ordering π of H such that $MCSLB(H, \pi)$ is as large as possible is NP-hard [16]. Using graph minor theory, one can show that the fixed parameter variant of this problem is tractable [16], but due to its non-constructive nature and huge constants hidden in the O-notation, this is only of theoretical interest.

8 Using Improved Graphs

A technique that has proved itself to be very powerful is the use of *improved graphs*, as introduced by Clautiaux et al. [17]. We start with a technical but important lemma.

Lemma 23 (See [7, 17]) *Let $G = (V, E)$ be a graph of treewidth at most k. Let v and w be two non-adjacent vertices in G. Suppose there are at least $k + 1$ vertex disjoint paths from v to w. Then the graph obtained by adding $\{v, w\}$ to G, $G' = (V, E \cup \{v, w\})$ has treewidth at most k.*

Lemma 23 generalises the following lemma, where we only look to common neighbours.

Lemma 24 (Bodlaender [5]) *Let $G = (V, E)$ be a graph of treewidth at most k. Let v and w be two non-adjacent vertices in G. Suppose there are at least $k+1$ vertices that are adjacent to v and to w. Then the graph obtained by adding $\{v, w\}$ to G, $G' = (V, E \cup \{v, w\})$ has treewidth at most k.*

Consider the following procedure, given a graph $G = (V, E)$ and an integer k. While there are non-adjacent vertices v and w that have at least $k + 1$ common neighbours in G, add the edge $\{v, w\}$ to G. Repeat this step till no such pair exists. Call the resulting graph the $(k + 1)$-neighbour improved graph.

Similarly, the $(k + 1)$-path improved graph of G is the graph obtained by repeatedly adding edges $\{v, w\}$ between non-adjacent vertices v and w that have at least $k + 1$ vertex disjoint paths between them. One can test in $O(k(|V| + |E|)$ time if a given pair has at least $k + 1$ vertex disjoint paths by using network flow techniques, see e.g., [32, Chapter 9] or [1].

From Lemmas 23 and 24 and induction, we obtain the following result.

Theorem 25 *Let G_n be the $(k+1)$-neighbour improved graph of graph $G = (V, E)$, and let G_p be the $(k+1)$-path improved graph of G. Then the treewidth of G is at most k, if and only if the treewidth of G_n is at most k, if and only if the treewidth of G_p is at most k.*

Based upon Theorem 25, Clautiaux et al. [17] give the LBN and LBP algorithm. This algorithm uses another treewidth lower bound algorithm (called in the pseudo-code in Algorithm 4 algorithm X) as a subroutine. In the original description by Clautiaux et al. [17], the degeneracy (MMD) is used, but one can actually use any treewidth lower bound algorithm at the place of X. For instance, using MMD+ instead of MMD will usually give much better lower bounds, while the extra cost in time is mainly due to the fact that more iterations will be done because of a better bound.

Algorithm 4 LBN(X) (Graph G)

```
low = X($G$)
repeat
  $H$ = the (low+1)-neighbour improved graph of $G$
  lowH = X($H$)
  changed = false
  if lowH > low then
    low = low+1
    changed = true
  end if
until not(changed)
return low
```

In Algorithm 4, we see how the LBN methods works. Using different lower bound algorithms at the place of X, we obtain e.g., the LBN(MMD), LBN(MMD+), or LBN(MCSLB) algorithm. In each iteration of the algorithm, we compute a lower bound on the treewidth of the (low+1)-neighbour improved graph H. If the treewidth of H is larger than low, then, by Theorem 25, the treewidth of G is larger than low, and thus we increase the known lower bound for the treewidth of G (variable low) by one.

The LBP method is almost identical. The only difference is that we compute the (low+1)-path improved graph instead of the (low+1)-neighbour improved graph. This often gives better lower bounds, but at the cost of additional running time. Correctness of the LBP method follows in the same way as for the LBN method.

Theorem 26 (Clautiaux et al. [17]) *If X is an algorithm that given a graph G, finds a lower bound on the treewidth of G, then LBN(X) and LBP(X) also are algorithms, that, when given a graph G, find a lower bound on the treewidth of G.*

Bodlaender, Koster, and Wolle [16] invented a new heuristic in which the LBN and LBP methods are combined with contraction by alternating improvement steps with contractions. Algorithm 5 gives the pseudo-code for the LBN+ method; the LBP+ method is again similar but uses path improved graphs instead of neighbour improved graphs.

Algorithm 5 LBN+(X) (Graph G)

```
low = X($G$)
repeat
  changed = false
  $H$ = the (low+1)-neighbour improved graph of $G$
  while (X($H$) $\leq$ low) and $H$ has at least one edge do
    Take a vertex $v$ of minimum degree in $H$
    Take a neighbour $u$ of $v$ according to least-c strategy
    Contract the edge $\{u, v\}$ in $H$
    $H$ = the (low+1)-neighbour improved graph of $H$
  end while
  if X($H$) $>$ low then
    changed = true
    low = low+1
  end if
until not(changed)
return low
```

Again, we can combine the LBN+ and LBP+ heuristics with any other lower bound heuristic for treewidth; in this way, we can build the LBN+(MD), LBP+(MMD+), LBP+(MCSLB), etc. The LBN+ and LBP+ methods start with computing a lower bound low for the treewidth of G. A graph H is taken that is the the (low+1)-neighbour or (low+1)-path improved graph of G. Then, alternately, a contraction of an edge is done, and H is set to the (low+1)-improved graph of the graph obtained by this contraction. This is done until H has no edges, or when we find that the lower bound on the treewidth of the graph H at hand, found by heuristic X, is larger than our current lower bound low. In the latter case, we increase low by one, and repeat again the entire process but now with the higher value of low.

Correctness of the procedure relies on the following lemma.

Lemma 27 *Let $G = (V, E)$ be a graph. Let X be an algorithm, that given a graph G, gives a lower bound on the treewidth of G. At each point during the execution of LBN+(X)(G) or LBP+(X)(G), we have that*

(i). The treewidth of G is at least low.

(ii). If low equals the treewidth of G, then the treewidth of H is at most low or changed $\equiv$ true.

Proof: We show that the properties hold invariantly during the running of the algorithm. Clearly, this holds after we have set low = X(G), and set H to the (low+1)-neighbour (or path) improved graph of G (see Theorem 25).

Contracting an edge in H does not affect the first invariant; the treewidth of H cannot increase by it, so it keeps the second invariant true.

Setting H to its (low+1)-neighbour (or path) improved graph does again not affect the first invariant. The second invariant keeps to be true. Suppose the treewidth of G equals low. Then, by assumption, the treewidth of H before the improvement operation is at most low, and by Theorem 25, the treewidth of H after the improvement operation is at most low.

Now, consider a step where the test X($H >$ low yields true. If X($H >$ low, then the treewidth of H is larger than low, and hence, by the second invariant, the treewidth of G does not equal low; by the first invariant, the treewidth of G must be at least low $+1$, so indeed we can safely increase low by one. Thus, the first invariant stays true, while the second invariant becomes trivially true as changed is set to true. □

One can also consider more complicated schemes, where first, a fast algorithm is used to get the first lower bounds (e.g., LBN(MMD+)), then a slower algorithm is used (LBN+(MMD+)), and then even slower algorithms are used (e.g., LBP+(MMD+), and then LBP+(MCSLB)), each continuing with the best bound obtained by the previous algorithms.

9 Brambles

Many of the algorithms of the previous sections exploit in some way or another the density of the graphs — graphs with many edges give good lower bounds (as they, or subgraphs or minors of them have large degree vertices), and with improvement, we try to add more edges to the graphs. For some types of graphs, these algorithms appear to perform usually poorly. A typical example are the planar graphs. For instance, as each planar graph has a vertex of degree at most five, the contraction degeneracy of a planar graph is at most five. In this section, we discuss a different technique, that is targeted at graphs that are planar, or close to planar.

This technique was introduced by Bodlaender, Grigoriev, and Koster in [9]. It appears to do quite badly on several dense graphs, but gives often excellent results on planar graphs, and graphs related to planar graphs. The bramble heuristic from [9] is based upon an alternative characterisation of treewidth in terms of the maximum order of a bramble of the graph, by Seymour and Thomas [33]. We use the terminology introduced by Reed [29].

Definition 28 *Let $G = (V, E)$ be a graph. Two subsets of V are said to* touch *if they have a vertex in common or E contains an edge between them. Let us call a set $\mathcal{B}$ of mutually touching connected vertex sets a* bramble. *A subset of V is said to* cover *$\mathcal{B}$ if it meets every element of $\mathcal{B}$. The least number of vertices covering a bramble is its* order.

The heuristic is based upon the following theorem by Seymour and Thomas [33]. A shorter proof of this result was given by Bellenbaum and Diestel [4].

Theorem 29 (Seymour and Thomas [33]) *Let k be a non-negative integer, and G be a graph. The treewidth of G is at least k if and only if G contains a bramble of order strictly greater than k.*

A nice example is a bramble for the r by r grid: take the collection of all sets of vertices that consist of one row and one column (crosses). This set of r^2 crosses is a bramble of order r. A small variant gives a bramble of order $r+1$, and thus a lower bound that matches the exact treewidth of the grid. See e.g., Figure 1. A heuristic of Bodlaender, Grigoriev, and Koster [9] finds a kind of generalisation of this type of brambles for arbitrary graphs.

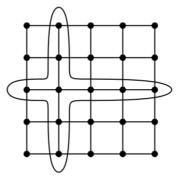


Figure 1: A 5 by 5 grid with a cross. A bramble of order 5 could consist of all 25 possible crosses

In the first step of the algorithm, we find a number of layers: disjoint sets of vertices, each inducing a connected subgraph of G. An arbitrary vertex is chosen as the only vertices in layer L_0. Then, repeatedly, given layer L_i, we first put in layer L_{i+1} all vertices that are adjacent to a vertex in layer L_i but do not belong to one of the first i levels, i.e., do not belong to $\bigcup_{j=1}^{i} L_i$. Then, we try to add vertices to layer L_i to make it induce a connected subgraph of G; e.g., we add to L_i the vertices on shortest paths between the different connected components of $G[L_i]$. If we made L_i connected, we proceed with building the next level; otherwise, all vertices in L_i and not in a level are added to level L_{i-1}. This is continued until all vertices belong to a level. We have now partitioned V into layers L_0, $\ldots$, L_r for some r, such that each $G[L_i]$ is connected.

Consider some α, β, $0 \leq \alpha < \beta \leq r$. Let $G_{\alpha,\beta}$ be the graph, obtained by contracting layers $L_0 \cup \cdots \cup L_\alpha$ to a single vertex v' and contracting layers $L_\beta \cup \cdots \cup L_r$ to a single vertex v''. $G_{\alpha,\beta}$ is a minor of G, so a lower bound on the treewidth of $G_{\alpha,\beta}$ is a lower bound on the treewidth of G. Using network flow techniques, we now find a maximum number of vertex disjoint paths from v' to v'' in $G_{\alpha,\beta}$. (See e.g., [1] for how this can be done efficiently in polynomial time.) Let $P_1, \ldots, P_c$ be these paths without the endpoints v' and v''.

Now, for each layer L_i, $\alpha < i < \beta$, and each path P_j, $1 \leq j \leq c$, we take the set $L_i \cup P_j$. As each layer is connected, each path intersects each layer, and a path is clearly connected, this set is connected. The bramble consists of all $(\beta - \alpha - 1)c$ combinations of layer and path, and one extra set consisting of vertex v' only. One can verify that this is indeed a bramble, and its order equals $\min(c+1, \beta - \alpha)$. So, the treewidth of $G_{\alpha,\beta}$ and hence of G is at least $\min(c+1, \beta - \alpha)$.

We can now try out all relevant pairs α, β, and then finding the number of vertex disjoint paths from v' to v'' and computing the corresponding treewidth lower bound as above, and finally taking the best bound obtained by the different combinations of α and β gives a lower bound for the treewidth. This process can be repeated with different choices for the start vertex v_0. We refer to [9] for more details. We also refer to [9] for a second heuristic, specially for planar graphs, that uses a variation of the principle.

10 Miscellaneous Techniques

10.1 As a Byproduct of Preprocessing

Several preprocessing algorithms yield as a byproduct a lower bound on the treewidth of the graph. For instance, there is a set of reduction rules with the property that for each graph G of treewidth at most three, there is a rule in the set that can rewrite G to an equivalent smaller instance [3, 14]. Thus, if we have a graph G, we can either rewrite it to the empty graph with these rules (and know the treewidth exactly), or we know that the treewidth of G is at least three. See for a further discussion [14] or [13].

10.2 Turning Exact Algorithms into Lower Bound Heuristics

The following simple technique can be used to turn exact algorithms for treewidth (or for many other problems) into lower bound heuristics. Many of the exact algorithms work with a known upper bound for the problem, and then find the best solution given this upper bound. Now, if we give instead of a real upper bound a conjectured lower bound α, the algorithm will find no solution at all, and we can conclude that the treewidth of the graph at hand is at least α. This technique can be used for example with a branch and bound algorithm (like the algorithm by Gogate and Dechter [20]) or with a dynamic programming algorithm. Recent work [8] has shown that this can help in several cases to get a better lower bound; some results have been reported in TreewidthLIB [40]. We plan to review exact algorithms for treewidth in [13].

10.3 Expansion Properties

Sunil Chandran and Subramanian [38] give another lower bound on the treewidth based upon an expansion property.

The open neighbourhood of a set of vertices $W \subseteq V$ in a graph $G = (V, E)$ is denoted $N(W) = \{x \in V \mid x \notin W \wedge \exists w \in W : \{w, x\} \in E\}$. For a graph $G = (V, E)$ and integer s, $1 \leq s \leq n$, we define $N_{\min}(G, s)$ to be the minimum of $|N(W)|$ over all sets of vertices $W \subseteq V$, $\frac{s}{2} \leq |W| \leq s$.

Theorem 30 (Sunil Chandran and Subramanian [38]) *For any graph $G = (V, E)$ and integer s, $1 \leq s \leq n$, the treewidth of G is at least $N_{\min}(G, s) - 1$.*

The result can be used as a lower bound heuristic by fixing some value s, and then computing $N_{\min}(G, s) - 1$. If s is a fixed integer, then this can be done in polynomial time. The most promising case may be taking $s = 3$. For larger values of s, probably the formula cannot be computed with sufficient efficiency.

11 Performance Analysis

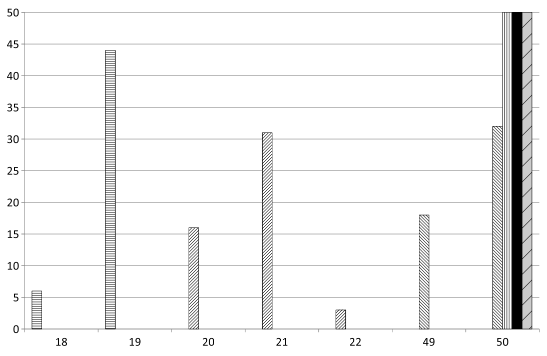
In this section we complement the discussion so far with a comptutational study. All presented results have been carried out on a Linux machine with 2.83 GHz Intel Core2 Quad CPU Q9550 and 8 GB RAM. The algorithms that have been evaluated have been implemented in C++ with use of the Boost Graph Library [36].

In most of the articles on treewidth lower bounds, computational results have been presented on selected instances from applications like probabilistic networks, vertex coloring, and frequency assignment. One of the drawbacks of such an approach is that for many graphs the optimal width is unknown, and thus only relative conclusions can be drawn. For this reason, we have have adapted in the first paper of this series [12] the approach of Shoikhet and Geiger [35] to test the algorithms on randomly generated partial-k-trees. In [12] we have tested the algorithms for all combinations of $n \in \{100, 200, 500, 1000\}$, $k \in \{10, 20\}$, and $p \in \{30, 40, 50\}$, where n is the number vertices, k the parameter of the partial-k-tree, and p the percentage of edges missing to be a k-tree. Where these graphs result in a good performance analysis of the upper bounds algorithms, they turn out to be of no value for a comparison of lower bound algorithms: the MMD+ algorithms provides a lower bound equal to k in all cases. Since k is an upper bound at the same time, no further improvements can be expected and the added value of algorithms like MCSLB or LBN+(MMD+) cannot be derived.

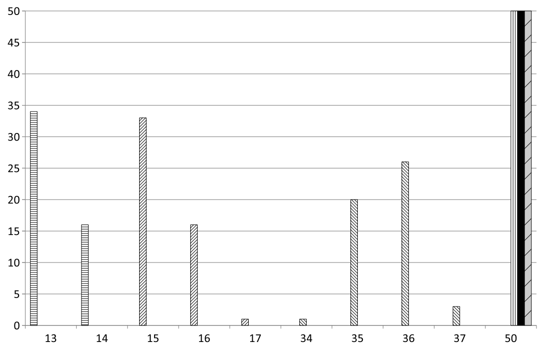
To arrive at a valuable performance analysis for selected lower bounds, we have adapted all three parameters. In this section, we present results for $n = 1000$, $k = 50$, and $p \in \{70, 80, 90\}$. Hence, 70%, 80%, or even 90% of the edges in a partial-50-tree of 1000 vertices have been deleted at random (i.e., the graphs contain 14618, 9745, or 4873 edges, respectively). For each case, we randomly generated 50 instances. The idea behind this setting is as follows. For all instances, $k = 50$ turned out to be the optimal width. For lower values of k or less missing edges, the lower bounds cannot be discriminated (even for $p = 60$ the MMD+ alggorithm generates the optimal width in all 50 cases). In case even more edges are deleted from the k-tree (e.g., $p = 95$), the optimal width drops below 50 several times and thus the results are more difficult to compare. Finally, for graphs with less vertices, the computation times of the faster algorithms are too small to be comparable.

For our computational comparison we selected the following algorithms: MMD, MCSLB, LBN(MMD), MMD+ (with *least-c* strategy), MCSLB+, and LBN+(MMD+). Other obvious choices like LBP+(MMD+) have been left out as the added value for these algorithms cannot be shown with this computational study (see below). This does not mean that these algorithms do not have an added value for other (particular) graphs.

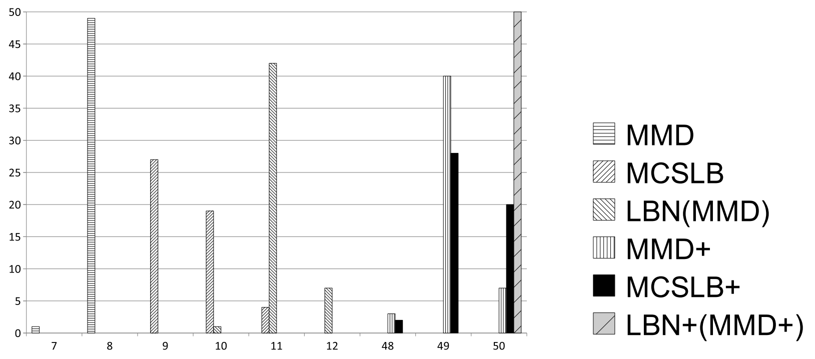
The detailed results are provided in Appendix A. Figure 2 summarizes the results by



(a) $n = 1000$, $k = 50$, $p = 70$



(b) $n = 1000$, $k = 50$, $p = 80$



(c) $n = 1000$, $k = 50$, $p = 90$

Figure 2: Histograms for selected lower bound algorithms

the means of histograms. Only the values reported at least once by one of the algorithms are presented on the x-axis. The y-axis shows the frequency this value is reported by a specific algorithm. The sum of the occcurences equals 50 for each algorithm.

From the results of algorithm LBN+(MMD+) we can conclude that the optimal width equals $k = 50$ for all instances. For $p = 70$ and $p = 80$, the same conclusion can be drawn by the results of the MMD+ or MCSLB+ algorithm, but these algorithms perform worse for $p = 90$ (with MCSLB+ slightly better than MMD+).

Figure 2 further shows that the quality of MMD and MCSLB decreases substantially when p increases. LBN(MMD) decreases most dramatically: for $p = 70$ it returns either 50 or 49, whereas for $p = 90$ it is only marginally better than MCSLB, reporting 10, 11, or 12.

In general, the contraction methods significantly outperform the algorithms without contraction. Where the performance of the algorithms without contraction decreases with increasing p, the contraction methods still perform very well.

The computation times in Appendix A show that MMD and MMD+ are incredible fast, even for these large instances. Both MCSLB and MCSLB+ run faster as the graph becomes more sparse. The same effect can be observed for LBN(MMD), but this can be explained by the smaller difference between MMD and LBN(MMD), hence less runs of

instance	queen8_8		queen12_12		tsp225	
algorithm	lb	cpu	lb	cpu	lb	cpu
MMD	21	0.00	33	0.00	4	0.00
MCSLB	21	0.02	33	0.13	5	0.14
LBN(MMD)	21	0.00	33	0.02	4	0.00
LBP(MMD)	23	0.60	37	26.31	8	73.76
MMD+	25	0.01	43	0.02	5	0.00
LBN(MMD+)	25	0.01	43	0.05	5	0.01
LBP(MMD+)	25	0.01	43	0.05	8	20.01
MCSLB+	25	0.19	44	1.98	5	0.75
LBN+(MMD)	26	0.14	43	2.25	5	0.13
LBN+(MMD+)	26	0.05	43	0.27	5	0.09
LBP+(MMD)	28	1.62	55	90.26	8	79.17
LBP+(MMD+)	28	0.61	55	50.02	8	20.07
bramble	2	0.01	2	0.03	10	1.73
planarbramble	-	-	-	-	9	0.06

Table 1: Lower bounds and computation times for selected graphs

the subalgorithm are undertaken. Similarly LBN+(MMD+) is substantially faster than MCSLB+ due to the good initial lower bound (roughly 3 seconds are needed for each run of the subalgorithm).

To show the added value of two further types of lower bounds, the path improvement and the bramble algorithm, we conclude this section with results for an extensive set of lower bounds for three particular graphs: queen8_8, queen12_12, and tsp225. The first two graphs origin from the DIMACS graph coloring challenge [19] and are examples of the n-queens puzzle, wheras the last one is a Delauney triangulation of a traveling salesman problem and has been used before by Hicks [21] to compute branchwidth. This latter graph is planar. Table 1 shows lower bounds and computation times for a variety of algorithms. From these results we can conclude that the more time consuming LBP algorithms indeed provide better lower bounds for the coloring graphs. Further, in the case of planar graphs, the bramble algorithms (a general version and a dedicated version for planar graphs exploiting an embedding in the plane) should be considered as they are likely to outperform all other considered algorithms in such cases (and are rather fast).

12 Conclusions

In this paper, we reviewed a number of algorithmic techniques to obtain lower bounds for the treewidth of graphs. For many graphs, taken from real-life applications, these techniques give good bounds, which in several cases match or come close to the best known upper bounds or exact treewidth of the graphs.

For many applications, good choices for lower bound heuristics are the LBN+(MMD+) and LBP+(MMD+) heuristics. The LBN+(MMD+) heuristic often is fast and gives good lower bounds; the LBP+(MMD+) heuristic is slower but gives usually still better lower bounds. For very large graphs, or when the heuristic is called frequently, e.g., as a subroutine of a branch and bound algorithm, then the LBN+(MMD+) and even more so the LBP+(MMD+) are probably too slow. A good choice then can be the MMD+ heuristic. Here, we see that the *least-c* strategy (contracting to the neighbour that has the smallest number of common neighbours) is usually the best choice as contraction strategy.

In some cases, these degree based methods fail heavily, e.g., when the input graphs are planar. In such cases, other methods, like computing the branchwidth exactly with the ratcatcher algorithm [21, 34] or using brambles can be tried.

There are several interesting questions for further research on treewidth lower bounds. For many of the heuristics, it probably is true that there are variants of the heuristics that give lower bounds of about equal quality, but that are also much faster. An example of this may be the LBP+(MMD+) heuristic. Suppose that after a few rounds of the algorithm, we have a lower bound for the treewidth of the original graph of k. Now, when we work with a graph whose MMD+ bound is much smaller than k, we might want to do a few contractions before we again compute an MMD+-bound. Also, in some cases we may want to skip a step that makes the improved graph in order to save some time — hopefully in such a way, that this does not affect the resulting lower bound. So, experimenting which combinations of techniques described in our overview paper give best combinations of quality of lower bounds and used time is worthwhile and interesting.

As in many cases, there are still large gaps between lower and upper bounds, having new and still better lower bound methods remains a good aim for research.

In this second paper of a series of overview papers on practical algorithms that compute or approximate the treewidth of graphs, we have seen several lower bound algorithms that in many cases work quite well. In the first paper in this series, we considered upper bound heuristics [12]; a third paper discussing exact algorithms and preprocessing methods is planned [13].

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Upper Saddle River, NJ, USA, 1993.

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

[3] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM Journal on Algebraic and Discrete Methods*, 7:305–314, 1986.

[4] P. Bellenbaum and R. Diestel. Two short proofs concerning tree-decompositions. *Combinatorics, Probability, and Computing*, 11:541–547, 2002.

[5] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

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

[7] H. L. Bodlaender. Necessary edges in k-chordalizations of graphs. *Journal of Combinatorial Optimization*, 7:283–290, 2003.

[8] 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 of the 14th Annual European Symposium on Algorithms, ESA 2006*, pages 672–683. Springer Verlag, Lecture Notes in Computer Science, vol. 4168, 2006.

[9] H. L. Bodlaender, A. Grigoriev, and A. M. C. A. Koster. Treewidth lower bounds with brambles. *Algorithmica*, 51:81–98, 2008.

[10] H. L. Bodlaender, T. Kloks, and D. Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM Journal on Discrete Mathematics*, 8(4):606–616, 1995.

[11] H. L. Bodlaender and A. M. C. A. Koster. On the Maximum Cardinality Search lower bound for treewidth. *Discrete Applied Mathematics*, 155(11):1348–1372, 2007.

[12] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208:259–275, 2010.

[13] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations III. Exact algorithms and preprocessing. Paper in preparation, 2010.

[14] 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.

[15] H. L. Bodlaender and T. Wolle. Contraction degeneracy on cographs. Technical Report UU-CS-2004-031, Institute for Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, 2004.

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

[17] F. Clautiaux, J. Carlier, A. Moukrim, and S. Négre. New lower and upper bounds for graph treewidth. In J. D. P. Rolim, editor, *Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms, WEA 2003*, pages 70–80. Springer Verlag, Lecture Notes in Computer Science, vol. 2647, 2003.

[18] J. Cole Smith, E. Ulusal, and I. V. Hicks. A combinatorial optimization algorithm for solving the branchwidth problem. Technical report, Rice University, 2010. URL: `http://www.caam.rice.edu/~ivhicks/`.

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

[20] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In D. M. Chickering and J. Y. Halpern, editors, *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence, UAI 2004*, pages 201–208. AUAI Press, 2004.

[21] I. V. Hicks. Planar branch decompositions I: The ratcatcher. *INFORMS Journal on Computing*, 17:402–412, 2005.

[22] I. V. Hicks, A. M. C. A. Koster, and E. Kolotoğlu. Branch and tree decomposition techniques for discrete optimization. In J. C. Smith, editor, *TutORials 2005*, INFORMS Tutorials in Operations Research Series, chapter 1, pages 1–29. INFORMS Annual Meeting, 2005.

[23] A. M. C. A. Koster, T. Wolle, and H. L. Bodlaender. Degree-based treewidth lower bounds. In S. E. Nikoletseas, editor, *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms, WEA 2005*, pages 101–112. Springer Verlag, Lecture Notes in Computer Science, vol. 3503, 2005.

[24] B. Lucena. *Dynamic Programming, Tree-Width, and Computation on Graphical Models*. PhD thesis, Brown University, Providence, RI, USA, 2002.

[25] B. Lucena. A new lower bound for tree-width using maximum cardinality search. *SIAM Journal on Discrete Mathematics*, 16:345–353, 2003.

[26] S. Ramachandramurthi. *Algorithms for VLSI Layout Based on Graph Width Metrics*. PhD thesis, Computer Science Department, University of Tennessee, Knoxville, Tennessee, USA, 1994.

[27] S. Ramachandramurthi. A lower bound for treewidth and its consequences. In E. W. Mayr, G. Schmidt, and G. Tinhofer, editors, *Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'94*, pages 14–25. Springer Verlag, Lecture Notes in Computer Science, vol. 903, 1995.

[28] S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM Journal on Discrete Mathematics*, 10:146–157, 1997.

[29] B. A. Reed. Tree width and tangles, a new measure of connectivity and some applications. In *Surveys in Combinatorics*, volume 241 of *LMS Lecture Note Series*, pages 87–162. Cambridge University Press, Cambridge, UK, 1997.

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

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

[32] A. Schrijver. *Combinatorial Optimization. Polyhedra and Efficiency.* Springer, Berlin, 2003.

[33] P. D. Seymour and R. Thomas. Graph searching and a minimax theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58:239–257, 1993.

[34] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[35] K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. In *Proceedings of the 15th National Conference on Artificial Intelligence, AAAI'97*, pages 185–190. Morgan Kaufmann, 1997.

[36] 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`.

[37] L. Sunil Chandran and C. R. Subramanian. A spectral lower bound for the treewidth of a graph and its consequences. *Information Processing Letters*, 87:195–200, 2003.

[38] L. Sunil Chandran and C. R. Subramanian. Girth and treewidth. *Journal of Combinatorial Theory, Series B*, 93:23–32, 2005.

[39] 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 Journal on Computing*, 13:566–579, 1984.

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

[41] T. Wolle. *Computational Aspects of Treewidth. Lower Bounds and Network Reliability.* PhD thesis, Faculty of Science, Utrecht University, Utrecht, the Netherlands, 2005.

[42] T. Wolle and H. L. Bodlaender. A note on edge contraction. Technical Report UU-CS-2004-028, Institute of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, 2004.

[43] T. Wolle, A. M. C. A. Koster, and H. L. Bodlaender. A note on contraction degeneracy. Technical Report UU-CS-2004-042, Institute of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, 2004.

A Computational Results

	MMD		MCSLB		LBN(MMD)		MMD+		MCSLB+		LBN+(MMD+)	
graph	lb	cpu	lb	cpu	lb	cpu	lb	cpu	lb	cpu	lb	cpu
70-01	19	0.00	20	6.32	49	35.69	50	0.14	50	147.00	50	7.34
70-02	19	0.00	20	6.46	49	39.87	50	0.14	50	144.32	50	7.42
70-03	18	0.00	20	6.45	49	48.62	50	0.13	50	145.25	50	7.14
70-04	18	0.00	20	6.45	50	62.16	50	0.14	50	144.36	50	7.49
70-05	19	0.01	22	6.46	50	33.25	50	0.13	50	141.81	50	7.27
70-06	19	0.01	21	6.41	50	37.10	50	0.13	50	142.28	50	7.38
70-07	19	0.00	22	6.45	50	36.59	50	0.14	50	147.09	50	7.23
70-08	19	0.00	21	6.53	50	36.76	50	0.13	50	142.82	50	7.38
70-09	19	0.01	21	6.49	50	37.51	50	0.13	50	143.21	50	7.23
70-10	19	0.00	21	6.42	49	35.81	50	0.13	50	142.73	50	7.22
70-11	19	0.01	21	6.60	50	33.69	50	0.13	50	141.78	50	7.39
70-12	19	0.00	21	6.39	50	38.86	50	0.14	50	142.71	50	7.27
70-13	19	0.00	20	6.45	50	37.99	50	0.14	50	143.35	50	7.43
70-14	19	0.01	21	6.48	50	37.76	50	0.13	50	144.36	50	7.25
70-15	18	0.00	20	6.54	49	58.32	50	0.14	50	143.20	50	7.29
70-16	19	0.00	21	6.45	50	35.23	50	0.13	50	142.41	50	7.25
70-17	18	0.00	20	6.47	49	54.39	50	0.14	50	147.18	50	7.45
70-18	19	0.01	21	6.45	50	36.41	50	0.13	50	146.35	50	7.23
70-19	19	0.00	21	6.67	49	34.38	50	0.13	50	143.93	50	7.27
70-20	19	0.01	21	6.46	49	36.24	50	0.13	50	143.27	50	7.31
70-21	19	0.01	21	6.46	50	34.95	50	0.13	50	145.91	50	7.33
70-22	19	0.01	21	6.48	50	35.75	50	0.13	50	142.61	50	7.33
70-23	19	0.00	21	6.51	49	34.75	50	0.14	50	141.42	50	7.23
70-24	19	0.01	20	6.60	50	36.64	50	0.13	50	142.29	50	7.36
70-25	19	0.01	21	6.48	50	39.35	50	0.13	50	141.78	50	7.28
70-26	19	0.00	21	6.54	50	35.36	50	0.13	50	141.25	50	7.41
70-27	19	0.00	21	6.43	49	32.81	50	0.14	50	145.25	50	7.31
70-28	19	0.00	20	6.47	49	33.00	50	0.14	50	139.07	50	7.28
70-29	19	0.00	20	6.41	50	36.84	50	0.13	50	144.70	50	7.32
70-30	19	0.01	20	6.43	50	35.13	50	0.13	50	146.63	50	7.38
70-31	19	0.00	21	6.40	50	34.44	50	0.14	50	140.88	50	7.27
70-32	19	0.00	20	6.45	50	35.02	50	0.13	50	145.15	50	7.22
70-33	19	0.00	21	6.37	49	36.28	50	0.14	50	140.58	50	7.15
70-34	19	0.01	21	6.48	50	34.78	50	0.13	50	140.23	50	7.31
70-35	19	0.00	21	6.45	50	36.85	50	0.13	50	146.91	50	7.39
70-36	19	0.00	21	6.47	50	36.98	50	0.13	50	144.22	50	7.31
70-37	19	0.01	21	6.45	50	33.45	50	0.13	50	143.24	50	7.21
70-38	19	0.00	21	6.46	50	37.87	50	0.13	50	142.43	50	7.31
70-39	19	0.00	22	6.44	50	39.38	50	0.14	50	141.66	50	7.32
70-40	19	0.00	20	6.43	49	36.04	50	0.13	50	152.76	50	7.27
70-41	18	0.00	21	6.43	49	58.77	50	0.14	50	143.03	50	7.34
70-42	19	0.00	20	6.45	50	37.25	50	0.13	50	142.01	50	7.41
70-43	19	0.00	20	6.44	49	36.03	50	0.13	50	140.75	50	7.34
70-44	18	0.01	21	6.56	49	54.21	50	0.13	50	143.34	50	7.43
70-45	19	0.01	21	6.50	50	36.41	50	0.13	50	143.41	50	7.32
70-46	19	0.01	21	6.61	49	33.86	50	0.13	50	143.12	50	7.22
70-47	19	0.00	21	6.49	50	34.64	50	0.14	50	145.43	50	7.34
70-48	19	0.00	20	6.44	50	32.19	50	0.14	50	143.06	50	7.40
70-49	19	0.00	21	6.48	50	38.90	50	0.13	50	142.60	50	7.47
70-50	19	0.01	21	6.49	49	38.71	50	0.13	50	145.13	50	7.32
avg.	18.9	0.00	20.7	6.47	49.6	38.47	50	0.13	50	143.61	50	7.32

Table 2: Results for $n = 1000$, $k = 50$, $p = 70$. CPU times are in seconds.

	MMD		MCSLB		LBN(MMD)		MMD+		MCSLB+		LBN+(MMD+)	
graph	lb	cpu	lb	cpu	lb	cpu	lb	cpu	lb	cpu	lb	cpu
80-01	13	0.00	15	5.24	36	23.19	50	0.09	50	96.26	50	5.52
80-02	13	0.00	15	5.19	35	20.50	50	0.09	50	97.36	50	5.37
80-03	13	0.01	16	5.19	36	21.06	50	0.08	50	99.03	50	6.00
80-04	14	0.01	16	5.22	35	14.41	50	0.08	50	103.20	50	5.42
80-05	13	0.01	15	5.20	35	22.39	50	0.08	50	97.42	50	5.55
80-06	13	0.01	15	5.22	35	23.02	50	0.08	50	96.11	50	5.47
80-07	13	0.01	15	5.20	35	26.64	50	0.08	50	97.21	50	5.34
80-08	13	0.00	15	5.38	35	22.79	50	0.09	50	98.40	50	5.56
80-09	14	0.00	16	5.23	36	15.38	50	0.08	50	97.42	50	5.49
80-10	13	0.00	15	5.23	35	24.71	50	0.08	50	96.80	50	5.50
80-11	13	0.00	15	5.32	35	22.66	50	0.09	50	98.18	50	5.41
80-12	13	0.00	15	5.19	36	28.44	50	0.08	50	99.43	50	5.46
80-13	14	0.00	16	5.13	37	14.60	50	0.08	50	95.71	50	5.41
80-14	14	0.00	16	5.15	35	14.61	50	0.09	50	97.52	50	5.39
80-15	13	0.01	16	5.20	37	23.51	50	0.08	50	97.97	50	5.41
80-16	14	0.01	15	5.64	36	15.67	50	0.08	50	98.12	50	5.34
80-17	13	0.01	15	5.30	34	24.62	50	0.08	50	97.38	50	5.43
80-18	13	0.00	15	5.21	36	23.46	50	0.09	50	99.05	50	5.39
80-19	14	0.01	15	5.21	35	12.55	50	0.08	50	95.48	50	5.39
80-20	13	0.00	17	5.23	35	26.05	50	0.08	50	96.78	50	5.38
80-21	14	0.01	15	5.29	36	14.73	50	0.08	50	98.73	50	5.42
80-22	13	0.00	15	5.23	36	27.61	50	0.09	50	97.18	50	5.56
80-23	14	0.01	16	5.16	36	15.05	50	0.08	50	97.60	50	5.48
80-24	13	0.00	15	5.26	35	25.73	50	0.08	50	95.46	50	5.41
80-25	13	0.00	15	5.21	36	27.18	50	0.09	50	99.97	50	5.44
80-26	14	0.01	15	5.30	35	14.23	50	0.08	50	97.88	50	5.40
80-27	13	0.00	15	5.17	36	25.37	50	0.09	50	96.56	50	5.46
80-28	14	0.01	16	5.16	35	14.75	50	0.08	50	98.28	50	5.41
80-29	14	0.00	15	5.21	37	15.19	50	0.08	50	98.08	50	5.42
80-30	14	0.01	15	5.17	36	15.91	50	0.08	50	99.18	50	5.44
80-31	13	0.00	15	5.22	35	26.25	50	0.09	50	96.49	50	5.42
80-32	13	0.00	15	5.15	36	27.26	50	0.09	50	98.65	50	5.44
80-33	13	0.00	16	5.25	36	25.03	50	0.09	50	97.14	50	5.44
80-34	13	0.01	16	5.26	36	24.75	50	0.08	50	100.11	50	5.37
80-35	14	0.00	16	5.28	36	13.92	50	0.08	50	108.39	50	5.51
80-36	13	0.01	15	5.16	36	24.43	50	0.08	50	100.52	50	5.46
80-37	13	0.00	16	5.24	35	26.24	50	0.08	50	96.72	50	5.37
80-38	13	0.01	16	5.18	36	20.87	50	0.07	50	99.34	50	5.42
80-39	13	0.01	15	5.18	36	23.35	50	0.08	50	104.23	50	5.38
80-40	13	0.00	15	5.21	35	22.49	50	0.09	50	96.71	50	5.33
80-41	14	0.00	15	5.18	35	14.31	50	0.09	50	98.48	50	5.43
80-42	13	0.00	15	5.24	35	20.74	50	0.09	50	98.44	50	5.54
80-43	14	0.00	15	5.51	36	13.92	50	0.08	50	97.90	50	5.41
80-44	13	0.00	15	5.26	36	22.10	50	0.09	50	96.39	50	5.40
80-45	13	0.00	15	5.16	36	23.42	50	0.09	50	96.67	50	5.42
80-46	13	0.00	15	5.15	35	24.56	50	0.08	50	97.69	50	5.46
80-47	13	0.00	16	5.13	36	22.90	50	0.08	50	96.61	50	5.46
80-48	14	0.00	16	5.16	36	14.17	50	0.08	50	98.87	50	5.39
80-49	13	0.00	16	5.19	36	23.42	50	0.09	50	96.85	50	5.45
80-50	13	0.00	15	5.22	36	27.01	50	0.08	50	96.51	50	5.43
avg.	13.3	0.00	15.4	5.23	35.6	21.14	50	0.08	50	98.13	50	5.44

Table 3: Results for $n = 1000$, $k = 50$, $p = 80$. CPU times are in seconds.

	MMD		MCSLB		LBN(MMD)		MMD+		MCSLB+		LBN+(MMD+)	
graph	lb	cpu	lb	cpu	lb	cpu	lb	cpu	lb	cpu	lb	cpu
90-01	8	0.00	10	3.76	12	2.34	49	0.04	49	46.98	50	6.48
90-02	7	0.01	9	3.74	11	9.09	49	0.04	50	46.60	50	6.43
90-03	8	0.01	10	3.81	11	2.90	49	0.04	50	47.91	50	6.20
90-04	8	0.01	9	3.73	11	2.64	49	0.04	50	47.38	50	6.78
90-05	8	0.00	10	3.72	11	2.60	49	0.04	49	46.83	50	6.28
90-06	8	0.00	10	3.79	11	2.41	49	0.03	49	47.53	50	6.52
90-07	8	0.01	9	3.71	11	2.83	49	0.04	50	47.51	50	6.39
90-08	8	0.00	10	3.74	10	2.98	49	0.04	49	49.72	50	6.43
90-09	8	0.01	9	3.77	11	2.28	48	0.04	48	47.29	50	9.28
90-10	8	0.01	10	3.72	11	2.75	49	0.04	50	47.14	50	6.35
90-11	8	0.01	10	3.69	12	2.96	48	0.04	49	47.56	50	9.27
90-12	8	0.00	9	3.75	11	2.69	49	0.03	50	47.22	50	6.26
90-13	8	0.00	10	3.74	11	2.27	49	0.04	49	47.09	50	6.19
90-14	8	0.00	10	3.76	11	2.82	49	0.04	49	47.55	50	6.42
90-15	8	0.01	9	3.66	11	3.39	50	0.03	50	46.89	50	3.21
90-16	8	0.01	9	3.72	12	2.82	49	0.04	49	47.60	50	6.48
90-17	8	0.00	10	3.68	11	2.63	50	0.04	49	46.89	50	3.19
90-18	8	0.01	9	3.70	11	1.99	49	0.04	50	47.79	50	6.37
90-19	8	0.01	9	3.73	11	2.35	49	0.04	49	46.71	50	6.45
90-20	8	0.00	9	3.68	12	2.64	50	0.04	50	47.10	50	3.27
90-21	8	0.01	9	3.84	11	2.54	49	0.04	48	47.93	50	6.40
90-22	8	0.00	10	3.75	11	3.05	49	0.04	49	47.64	50	6.38
90-23	8	0.01	10	3.77	11	1.92	49	0.03	50	47.47	50	6.34
90-24	8	0.01	10	3.68	11	3.26	49	0.04	49	48.28	50	6.30
90-25	8	0.01	10	3.71	11	2.37	49	0.03	49	48.07	50	6.24
90-26	8	0.01	9	3.76	11	2.64	49	0.04	49	50.53	50	6.33
90-27	8	0.01	9	3.73	11	2.61	49	0.04	49	48.03	50	6.52
90-28	8	0.00	9	3.65	11	2.68	49	0.04	49	46.33	50	6.41
90-29	8	0.01	10	3.70	11	2.64	49	0.03	49	47.03	50	6.44
90-30	8	0.00	9	3.70	11	2.83	50	0.04	50	46.76	50	3.28
90-31	8	0.00	9	3.76	11	2.36	48	0.04	50	47.32	50	9.53
90-32	8	0.01	9	3.76	11	2.32	50	0.04	49	47.11	50	3.24
90-33	8	0.01	11	3.76	11	2.68	50	0.04	49	47.33	50	3.27
90-34	8	0.00	10	3.70	11	2.22	49	0.04	50	46.77	50	6.30
90-35	8	0.00	9	3.70	11	3.05	49	0.04	49	47.03	50	6.37
90-36	8	0.00	9	3.66	12	2.81	49	0.04	49	46.55	50	6.33
90-37	8	0.00	9	3.85	11	2.53	49	0.04	50	47.54	50	6.42
90-38	8	0.00	10	3.73	12	3.27	49	0.04	50	46.31	50	6.22
90-39	8	0.00	9	3.76	11	2.50	49	0.04	49	47.34	50	6.40
90-40	8	0.01	9	3.80	11	1.82	49	0.04	49	52.49	50	6.44
90-41	8	0.00	9	3.71	11	2.19	49	0.04	49	46.98	50	6.25
90-42	8	0.01	9	3.77	12	2.49	49	0.04	50	47.22	50	6.35
90-43	8	0.01	11	3.74	11	3.31	49	0.04	50	47.17	50	6.34
90-44	8	0.00	11	3.68	11	3.14	49	0.04	49	46.61	50	6.28
90-45	8	0.01	9	3.73	11	2.98	50	0.03	50	47.38	50	3.32
90-46	8	0.01	11	3.76	11	2.34	49	0.04	49	46.75	50	6.51
90-47	8	0.01	10	3.75	11	3.18	49	0.03	49	46.77	50	6.44
90-48	8	0.00	9	3.77	11	2.41	49	0.04	50	47.41	50	6.46
90-49	8	0.00	10	3.80	11	2.74	49	0.04	49	46.98	50	6.34
90-50	8	0.00	9	3.71	11	2.53	49	0.04	50	46.81	50	6.33
avg.	7.98	0.01	9.54	3.74	11.1	2.78	49.1	0.04	49.4	47.42	50	6.12

Table 4: Results for $n = 1000$, $k = 50$, $p = 90$. CPU times are in seconds.