Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

HANS L. BODLAENDER
ALEXANDER GRIGORIEV
ARIE M.C.A. KOSTER

Treewidth Lower Bounds with Brambles

ZIB-Report 05–54 (December 2005)

Treewidth Lower Bounds with Brambles*

Hans L. Bodlaender† Alexander Grigoriev‡ Arie M. C. A. Koster§

Abstract

In this paper we present a new technique for computing lower bounds for graph treewidth. Our technique is based on the fact that the treewidth of a graph G is the maximum order of a bramble of G minus one. We give two algorithms: one for general graphs, and one for planar graphs. The algorithm for planar graphs is shown to give a lower bound for both the treewidth and branchwidth that is at most a constant factor away from the optimum. For both algorithms, we report on extensive computational experiments that show that the algorithms give often excellent lower bounds, in particular when applied to (close to) planar graphs.

1 Introduction

Motivation. In many applications of the notion of treewidth, it is desirable that we can compute tree decompositions of small width of given graphs. Unfortunately, finding a tree decomposition of minimum width and determining the exact treewidth are NP-hard; see [3]. Much research has been done in recent years on the problem to determine the treewidth of the graph: this includes a faster exponential time algorithm [20], a theoretically optimal but due to the large constant factor hidden in the O-notation impractical linear time algorithm for the fixed parameter case [5], a polynomial time algorithm for graphs with polynomially many minimal separators [13], a branch and bound algorithm [22], preprocessing methods [9, 10], upper bound heuristics [2, 15, 27], and lower bound heuristics [11, 14, 28, 29, 31]. An overview with many references can be found in [7].

In this paper, we focus on lower bound methods for treewidth. Lower bound algorithms are interesting and useful for a number of different reasons. When running a branch and bound algorithm to compute the treewidth of a graph, see e.g. [22], a good lower bound

*This work was partially supported by the Netherlands Organisation for Scientific Research NWO (project *Treewidth and Combinatorial Optimisation*) and partially by the DFG research group "Algorithms, Structure, Randomness" (Grant number GR 883/9-3, GR 883/9-4).

†Department of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. Email: `hansb@cs.uu.nl`

‡Department of Quantitative Economics, University of Maastricht, P.O. Box 616, 6200 MD Maastricht, The Netherlands. Email: `a.grigoriev@ke.unimaas.nl`

§Zuse Institute Berlin (ZIB), Takustraße 7, D-14195 Berlin-Dahlem, Germany. Email: `koster@zib.de`

helps to quickly cut off branches. Lower bounds inform us on the quality of upper bounds. Also, a high lower bound can tell that we should not aim for a solution of a problem on a certain graph instance with treewidth techniques: Suppose we decide we want to solve a certain problem on a given graph with a dynamic programming algorithm on a tree decomposition. If we have a large lower bound for the treewidth of that graph, we know in advance that this dynamic programming algorithm will use much time and memory, and hence we should direct our attention to trying different methods. Finally, better lower bounds for treewidth sometimes help to obtain further reductions in the size of graphs obtained by preprocessing [10].

In recent years, several treewidth lower bound methods have been found and evaluated. A trivial lower bound for the treewidth is the minimum degree of a vertex. Better lower bounds are obtained by looking at the minimum degrees of induced subgraphs or of graphs obtained by contractions, see [11]. Often slightly better than the minimum degree is a lower bound by Ramachandramurthi [31], which is (for non-complete graphs) the minimum over all pairs of non-adjacent vertices v, w, of the maximum degree of v and w. Another lower bound, found by Lucena [29] and analyzed in [8], is based on maximum cardinality search. Combining these bounds with contractions gives often considerable improvements [11, 28]. Further improvements can be obtained by using these techniques in combination with a method introduced by Clautiaux et al. [14], based upon adding edges between vertices that have many common neighbors or disjoint paths between them [12].

The experiments carried out in [11, 28] show that for several graphs, the existing lower bound methods give good bounds that are often close and in several cases equal to the actual treewidth. However, there are classes of graphs where each of these methods yields a rather small lower bound that is far away from the real treewidth. For instance, this holds for planar graphs or graphs that are in a certain sense close to being planar, e.g., graphs obtained by taking the union of a small number of TSP-tours on a point set in the plane [16]. The reason that the above mentioned techniques appear to fail for these graphs mostly is due to the fact that all these methods in a certain sense are (minimum) degree-based, and (close to) planar graphs always have vertices of small degree, cf. [41].

For planar graphs, Seymour and Thomas [38] presented an $O(n^2)$ time algorithm computing a minimum branchwidth. Branchwidth of a graph is a lower bound for the treewidth, and moreover, the branchwidth of a graph approximates the treewidth within a factor of 3/2, see Robertson and Seymour [34]. Unfortunately, the algorithm of Seymour and Thomas requires a substantial, although polynomial, amount of memory, so it runs out of memory even for the medium size graphs (from 2000 nodes), see [24, 25]. In the later two papers, Hicks proposed another algorithm that at cost of slowing down the algorithm of Seymour and Thomas makes it "*memory friendly*". The algorithm of Hicks runs in time $O(n^3)$ and can deal with graphs up to 15000 nodes.

Since the known algorithms for a treewidth lower bound are either slow, or not memory friendly, or provide a bad quality bound, we were searching for a new method using a different principle that works well for graphs that are planar or close to planar. In this paper, we present such a different method, based on the notion of *bramble* (for the first time, brambles appeared in [37] with the name *screens*).

Notations and preliminaries. Throughout this paper, $G = (V, E)$ denotes an undirected simple graph. We use many of the standard graph theoretic notions in the usual way, like path, connected subgraph, etc. For a subset $S \subseteq V$ of the vertices, we denote with $N(S)$ the neighbors of S that are outside S, i.e., $N(S) = \{w \in V \setminus S : \{v, w\} \in E, v \in S\}$. With $G[S]$ we denote the subgraph of G induced by $S \subseteq V$. Instead of $G[V \setminus S]$ we also write $G \setminus S$.

The *vertex connectivity* of non-adjacent vertices $v, w \in V$ in G is defined as the minimum size of a vertex set $S \subset V$ such that v and w are in different components of $G \setminus S$. By Menger's Theorem [30], the number of vertex disjoint paths between v and w equals the vertex connectivity. The vertex connectivity of a pair v, w and the vertex disjoint paths can be computed by maximum flow techniques, see e.g. [1].

The notions of tree decomposition and treewidth were first defined by Robertson and Seymour [33]. Several other, equivalent notions, have been proposed by many different authors, see e.g., [6].

Definition 1.1 *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 a 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.*

A graph H is a *minor* of G, if H can be obtained from G by a series of zero or more vertex deletions, edge deletions, and edge contractions. It is well known that if H is a minor of G, then the treewidth of H is at most the treewidth of G.

A notion closely related to treewidth is the branchwidth of a graph, also introduced by Robertson and Seymour [34, 35].

Definition 1.2 *A* branch decomposition *of a graph $G = (V, E)$ is a pair $(T = (I, F), \sigma)$, with T a ternary tree (a tree where every non-leaf node has degree 3) with $|E|$ leaves, and σ a bijection from E to the set of leaves of T.*

The order *of an edge $f \in F$ is the number of vertices $v \in V$, for which there exist adjacent edges $\{v, w\}, \{v, x\} \in E$, such that the path in T from $\sigma(v, w)$ to $\sigma(v, x)$ uses f.*

The width *of branch decomposition $(T = (I, F), \sigma)$, is the maximum order over all edges $f \in F$. The branchwidth of a graph G, $bw(G)$, is the minimum width over all branch decompositions of G.*

Robertson and Seymour [34] proved that $\max\{bw(G), 2\} \le tw(G)+1 \le \max\{\lfloor 3bw(G)/2 \rfloor, 2\}$. One easily can convert a branch decomposition into a tree decomposition such that the respective widths fulfil these inequalities.

We next give the definitions of bramble and related notions, following the terminology of Reed [32].

Definition 1.3 *Let $G = (V, E)$ be a graph. Two subsets $W_1, W_2 \subseteq V$ are said to* touch *if they have a vertex in common or E contains an edge between them ($W_1 \cap W_2 \neq \emptyset$ or there is an edge $\{w_1, w_2\} \in E$ with $w_1 \in W_1$, $w_2 \in W_2$). A set $\mathcal{B}$ of mutually touching connected vertex sets is called a* bramble. *A subset of V is said to* cover *$\mathcal{B}$ if it it is a hitting set for $\mathcal{B}$ (i.e., a set which intersects every element of $\mathcal{B}$.) The* order *of a bramble $\mathcal{B}$ is the minimum size of a hitting set for $\mathcal{B}$. The* bramble number *of G is the maximum order of all brambles of G.*

The relationship between the bramble order and the treewidth was obtained by Seymour and Thomas [37]:

Theorem 1.4 (Seymour and Thomas [37]) *Let k be a non-negative integer. A graph has treewidth k if and only if it has bramble number $k + 1$.*

For a short proof of this theorem we refer to Bellenbaum and Diestel [4].

Corollary 1.5 *Given a bramble $\mathcal{B}$ of order k, $tw(G) \geq k - 1$.*

So, finding a high order bramble immediately implies getting a good lower bound for the treewidth. Unfortunately, determining the order of a bramble, is also NP-hard; it follows directly from the NP-hardness of the minimum hitting set problem [21] by taking a complete graph G and the collection of subsets as the bramble.

A class of graphs for which the bramble of maximum order can be constructed easily is the class of grid graphs. It is folklore that an r by r grid has treewidth r; see e.g., Figure 1 for a bramble of order $r + 1$. We have one set that contains all vertices on the bottom row, one set that contains all vertices on the last column except the last one, and $(r - 1)^2$ *crosses*, each consisting of the first $r - 1$ vertices of one of the first $r - 1$ rows and the first $r - 1$ vertices of one of the first $r - 1$ columns. A set that covers the bramble must contain at least $r - 1$ vertices to cover the crosses, and one vertex in each of the other two sets.

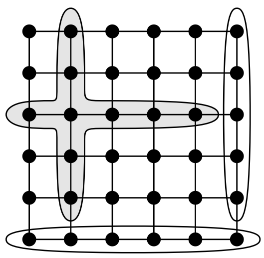


Figure 1: Illustrating a bramble of a grid

In this paper, we therefore restrict our search to brambles for which the order can be computed easily, due to the construction of the bramble. In Section 2 we construct brambles for general graphs; and in Section 3 a more sophisticated, though fast, algorithm for planar graphs is presented. The latter algorithm is a constant approximation algorithm for both treewidth and branchwidth of planar graphs. In Section 4, we report on computational results, showing the effectiveness of the brambles as lower bound for treewidth in those cases that the graph is (close to) planar. We finish the paper with some concluding remarks.

2 Brambles in general graphs

In this section, we present Algorithm $\mathcal{A}_1$. Algorithm $\mathcal{A}_1$ gets as input an arbitrary undirected graph G, finds several brambles of G or of minors of G, and outputs the best treewidth lower bound found this way.

First, in an initialisation step, the set of vertices is split in a number of *level sets*, $V_0, \ldots, V_R$ (with R defined within the initialisation). For simplicity, the level sets are called by their levels. In the basic step, a number of brambles is constructed, and their order is computed. The maximum order encountered is recorded and returned at the end.

Before we can give a detailed description of the algorithm, we have to introduce some additional notation. Given a subset $S \subseteq V$ of the vertices with $G[S]$ not connected, we call a set T a *connectivity closure* of S in G if $G[S \cup T]$ is connected. If G is not connected, there does not exist a connectivity closure of S in G.

A, not necessarily minimal, connectivity closure is provided by the following procedure: Let $T := \emptyset$ in the beginning. Construct a bidirectional graph $D = (V, A)$ with two arcs for every edge in E. For $(v, w) \in A$, define a length of 0 if $v, w \in S \cup T$ and 1 otherwise. Now, compute the shortest path from an arbitrary vertex $v \in S \cup T$ to all other vertices in S. The distance between two vertices is either zero or at least two. If the length of a path between v and w is at least two, v and w are in two different components of $G[S \cup T]$, and the distance minus one is the minimum number of vertices required to connect both components. Now, let w be a vertex for which the distance is minimum among those with distance at least two. We add the vertices on the (v, w)-path to T and restart the procedure from the construction of D, until no distance between vertices of $S \cup T$ exceeds zero.

Now, we are ready to define the algorithm.

Algorithm $\mathcal{A}_1$

Initialisation.

1. Take an arbitrary vertex r in V. Define $V_0 := \{r\}$. Set $k := 1$.

2. Let $V'_k = N(V_{k-1}) \setminus \{\bigcup_{i=0}^{k-1} V_i\}$ be the vertices adjacent from V_{k-1} that are not part of the existing subsets. If $G[V'_k]$ is connected, then $V_k := V'_k$. Otherwise, we search for a connectivity closure V''_k of V'_k in $G \setminus \{\bigcup_{i=0}^{k-1} V_i\}$. If such a closure exists, let $V_k := V'_k \cup V''_k$. If there is no connectivity closure for the set V'_k, then redefine $V_{k-1} := V \setminus \{\bigcup_{i=0}^{k-2} V_i\}$ and go to step 3 of initialisation. In all other cases, set $k := k + 1$ and return to the beginning of step 2.

3. Set $R := k$. Add to the graph a dummy vertex q; connect all vertices from the level $R - 1$ to q; and define $V_R := \{q\}$.

4. Initialize an integer variable *bestlow* to -1.

Basic Step. For all $1 \le i \le j \le R-1$ do the following.

1. Let $G_{i,j}$ be the graph resulting from G after contraction of the first i levels $V_0, \ldots, V_{i-1}$ into vertex s_i and contraction of the last $R-j$ levels $V_{j+1}, \ldots, V_R$ into vertex t_j.

2. Let c_{ij} be the vertex connectivity of the pair (s_i, t_j) in $G_{i,j}$. Moreover, let P_ℓ, $\ell = 1, \ldots, c_{ij}$, denote the internal vertices of c_{ij} vertex disjoint paths between s_i and t_j.

3. Define $\mathcal{B}_{i,j}$ as follows. Let $\{s_i\}$ be an element of $\mathcal{B}_{i,j}$. For all $i \le k \le j$ and $1 \le \ell \le c_{i,j}$ let the set $V_k \cup P_\ell$ be an element of $\mathcal{B}_{i,j}$.

4. Set *bestlow* to the maximum of *bestlow* and $\min\{c_{i,j}, j-i+1\}$.

Output. Output $LB_1 = bestlow$. STOP.

Notice, that the procedure described in step 2 of the initialisation is a classical Breadth-First-Search (BFS) extended to taking connectivity closures. Moreover, note that for $1 \le i \le j \le R-1$, the vertices s_i and t_j are not adjacent in $G_{i,j}$ by definition of the level sets, and thus the vertex connectivity is defined for these vertex pairs.

Theorem 2.1 *Set* $\mathcal{B}_{i,j}, 1 \le i \le j \le R-1$, *is a bramble of* $G_{i,j} \setminus \{t_j\}$. *The order of this bramble equals* $\min\{c_{i,j}+1, j-i+2\}$.

Proof: Take any pair i, j such that $1 \le i \le j \le R-1$ and consider the corresponding graph $G_{i,j}$. Let us check whether all conditions in the definition of brambles are satisfied for the set $\mathcal{B}_{i,j}$.

First of all, let us verify that each element of $\mathcal{B}_{i,j}$ induces a connected set. From initialisation we know that each level is a connected set. The paths P_ℓ constructed in the basic step of $\mathcal{A}_1$ are clearly connected and each of those paths crosses all levels in $G_{i,j}$. Hence a union of any level and any path forms a connected set.

Secondly, all elements of $\mathcal{B}_{i,j}$ are mutually touching: all unions of level and path are touching $\{s_i\}$, and again all paths from the basic step in $\mathcal{A}_1$ pass through all levels in $G_{i,j}$. Therefore, $\mathcal{B}_{i,j}$ is a bramble for $G_{i,j}$.

Let $C \subset V$ be a vertex set that determined the vertex connectivity of the pair s_i, t_j. Clearly, $C \cup \{s_i\}$ forms a cover for the constructed bramble. A set of representative vertices, one from each level k, $i \le k \le R-j$, together with s_i also form a cover for the constructed bramble. Thus, the order of the bramble is at most $\min\{c_{i,j}+1, j-i+2\}$. Since the levels are non-intersecting, the paths constructed at the basic step of $\mathcal{A}_1$ are vertex disjoint, and the bramble contains all combinations of these levels and paths, the order of the bramble is at least $\min\{c_{i,j}+1, j-i+2\}$, which completes the proof. $\square$

Note that, although $G_{i,j}$ is a minor of G if $j < R-1$, the bramble cannot be extended with subset $\{t_j\}$ since s_i and t_j are not touching.

Corollary 2.2 *For each graph* G, $LB_1 \le tw(G)$.

Proof: By Theorem 2.1, each time *bestlow* is increased, it is set to the order of a bramble of a graph $G_{i,j}$ minus one. By Corollary 1.5, this value is a lower bound on the treewidth of $G_{i,j}$, and as $G_{i,j}$ is a minor of G, this is also a lower bound on the treewidth of G. $\square$

Theorem 2.3 *Algorithm $\mathcal{A}_1$ can be carried out in $O(n^3m)$ time on a graph G with n vertices and m edges.*

Proof: Computing the partition of V into the level sets requires at most R constructions of a connectivity closure. The proposed procedure for this takes at most n computations of shortest paths from a source to all other nodes in the graph, each taking $O(n^2)$ time in a straightforward implementation of Dijkstra's algorithm, see [1] pp. 108-112. Thus, the initialisation requires at most $O(Rn^3)$ time.

In the basic step of the algorithm we compute c vertex disjoint (s,t)-paths in a graph, that can be done in $O(nm)$ time; see [1] pp. 273-277. Together with enumeration over all possibilities for i and j, it brings the time complexity of the basic step up to $O(R^2nm)$. Since $R \leq n$, we have that the total running time of $\mathcal{A}_1$ is at most $O(n^3m)$, as required. $\square$

At cost of an additional multiplicative factor of n we can find a root vertex r for the BFS that provides the best lower bound for the treewidth, cf. Section 4.

3 Brambles in planar graphs

Algorithm $\mathcal{A}_1$ can be significantly improved in running time when the input graph is restricted to be planar. Given an embedding of $G_{i,j}\backslash\{s_i, t_j\}$, the vertex disjoint paths P_ℓ can be viewed as layers in the graph, with the remaining vertices laying between (below/above) these layers. Since every level set V_i is connected as well and all paths intersect with V_i, the vertices in between the layers can be contracted either to the layer above or below. If we further contract vertices on a path that belong to the same level set V_i, the resulting graph contains a c_{ij} by $j-i+1$ grid, and hence the treewidth of $G_{i,j}$ is at least $\min\{c_{ij}, j-i+1\}$. The need for a root vertex r, however, causes that for grid graphs the treewidth may not be reached by Algorithm $\mathcal{A}_1$.

In this section we present a different algorithm that searches for grid minors in planar graphs more directly. Algorithm $\mathcal{A}_2$ below finds brambles in several minors of a connected planar graph G. In the initialisation step, we partition the vertices on the exterior face in *North*, *East*, *West*, and *South*. In the basic step, the algorithm builds grid minors of G for which we know how the maximum order bramble looks like. Algorithm $\mathcal{A}_2$ outputs the largest lower bound due to the grid minors and corresponding brambles it has met.

Algorithm $\mathcal{A}_2$

Input. A planar embedding of G with no edge crossings. It is well known that such an embedding can be constructed in linear time (e.g., the $O(n+m)$ algorithm by Hopcroft and Tarjan [26]).

Initialisation. Let F be the exterior face of the embedding and let f be the length of a single closed clockwise walk along F. Since G is a simple graph, $f \geq |F| \geq 3$. Let $North$, $East$, $South$, and $West$ be four paths on F (possibly atomic if F is a simple cycle on 3 vertices) being sequential parts of a closed clockwise walk along F such that $North$ has one common endpoint with $East$, $East$ has one common endpoint with $South$, $South$ with $West$, and finally $West$ with $North$. Moreover, let the lengths of these four paths be roughly the same, i.e., the length may vary by at most one vertex being either $\lfloor f/4 \rfloor + 1$ or $\lceil f/4 \rceil + 1$. Notice that such paths always exist and can be found in linear time. We add in the exterior four dummy vertices $\mathcal{N}, \mathcal{E}, \mathcal{S}$, and $\mathcal{W}$ and connect them to all vertices in $North$, $East$, $South$, and $West$ respectively. Further in the paper we always refer to the vertices incident to $\mathcal{N}, \mathcal{E}, \mathcal{S}, \mathcal{W}$ as to $North$, $East$, $South$ and $West$, respectively; and we denote $Dummy = \{\mathcal{N}, \mathcal{E}, \mathcal{S}, \mathcal{W}\}$. Finally, we set integer variable *bestlow* to -1.

Basic step. We view the algorithm as a rooted search tree with a root corresponding to the graph constructed in the initialisation. At each node of the search tree we perform the following steps.

1. Given is a node i in the tree and the planar graph $G_i = (V_i, E_i)$ associated with this node. Let c_i be the vertex connectivity of $(\mathcal{N}, \mathcal{S})$ in graph $G_i \setminus \{\mathcal{E}, \mathcal{W}\}$ and d_i the vertex connectivity of $(\mathcal{W}, \mathcal{E})$ in $G_i \setminus \{\mathcal{N}, \mathcal{S}\}$. Find c_i vertex disjoint paths connecting $\mathcal{N}$ and $\mathcal{S}$, and d_i vertex disjoint paths connecting $\mathcal{W}$ and $\mathcal{E}$. Clearly, $G_i \setminus Dummy$ contains a $c_i \times d_i$ grid as a minor, and therefore it contains also a $b_i \times b_i$ grid as a minor, where $b_i = \min\{c_i, d_i\}$. Create an order $b_i + 1$ bramble $\mathcal{B}_i$ for this $b_i \times b_i$ grid as described in Section 1. Set *bestlow* to the maximum of *bestlow* and b_i.

2. If $b_i = c_i$, we create the child nodes of i as follows. Let $C_i \subset V_i$ be a vertex set determining the vertex connectivity of $(\mathcal{N}, \mathcal{S})$ in $G_i \setminus \{\mathcal{E}, \mathcal{W}\}$. Further in the text we refer to such a set as to *separator*. Separator C_i specifies at least two components in $G_i \setminus (C_i \cup \{\mathcal{E}, \mathcal{W}\})$: one containing $\mathcal{N}$ and another containing $\mathcal{S}$. Notice that there can be other components eventually separated by C_i from both $\mathcal{N}$ and $\mathcal{S}$. For each component we create a child node in the search tree and construct the graph associated with this child node as follows.

   Let G_i^q be a component in $G_i \setminus (C_i \cup \{\mathcal{E}, \mathcal{W}\})$. Consider the exterior face X of $G_i^q \setminus Dummy$. Now, like in the initialisation, we define four paths $North$, $East$, $South$, and $West$ in a single closed clockwise walk along X. If a vertex $v \in X$ belongs to $North$ in G_i, let this vertex belong to $North$ in G_i^q as well. In the similar way we leave the vertices which are already assigned to $East$, $South$, and $West$ in G_i in the corresponding paths in G_i^q.

   Yet unassigned vertices in X we assign to the paths in such a way that, again, the paths (possibly atomic) become sequential parts of a closed clockwise walk along X such that $North$ has one common endpoint with $East$, $East$ has one common endpoint with $South$, $South$ with $West$, and $West$ with $North$. Finally, we connect

$\mathcal{N}$ to $North$, $\mathcal{E}$ to $East$, $\mathcal{S}$ to $South$, and $\mathcal{W}$ to $West$. Let the resulting graph be a graph associated with a child node of i corresponding to component G_i^q.

3. If $d_i < c_i$, we similarly create the child nodes with respect to the vertex connectivity of $(\mathcal{E}, \mathcal{W})$.

4. We recurse on the child nodes unless the number of non-dummy vertices in the graph associated with the node becomes less or equal to the current value of *bestlow*.

Output. Output $LB_2 = bestlow$. STOP.

Theorem 3.1 *The lower bound LB_2 on the treewidth of a planar graph can be obtained by Algorithm $\mathcal{A}_2$ in time $O(n^2 \log n)$.*

Proof: We already observed that for any node i in the search tree, b_i is a side size of a square grid minor in G. Since the treewidth of a square grid is its side size we directly have that $bestlow = \max_i b_i$ is a lower bound on the treewidth of G.

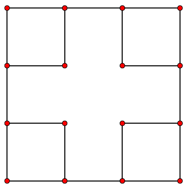
Notice that the initialisation requires only linear time. In the basic step of the algorithm we compute two vertex connectivities in a planar network, which can be done in $O(n \log n)$ time; see Theorem 8.8 in [1], p. 265. Then we create the child nodes in the search tree basically determining the connected components in a graph which takes linear time. Therefore, the basic step of the algorithm requires $O(n \log n)$ time.

We perform the basic step for all nodes of the search tree. Let us argue that the number of nodes in this tree is at most n. Consider leaf nodes of the tree. By construction, the vertex sets of the graphs associated with all leaf nodes are mutually disjoint. Now, consider the immediate parent node of some nodes. The graph associated with this parent node contains a vertex set of G (determining the vertex connectivity of two dummy vertices at a certain basic step of the algorithm) which is removed from all child nodes. Therefore, moving up to the root, in each parent node we find at least one vertex which is not present in any of its children. Since the number of vertices in V is n, we conclude, that the search tree has size at most n.

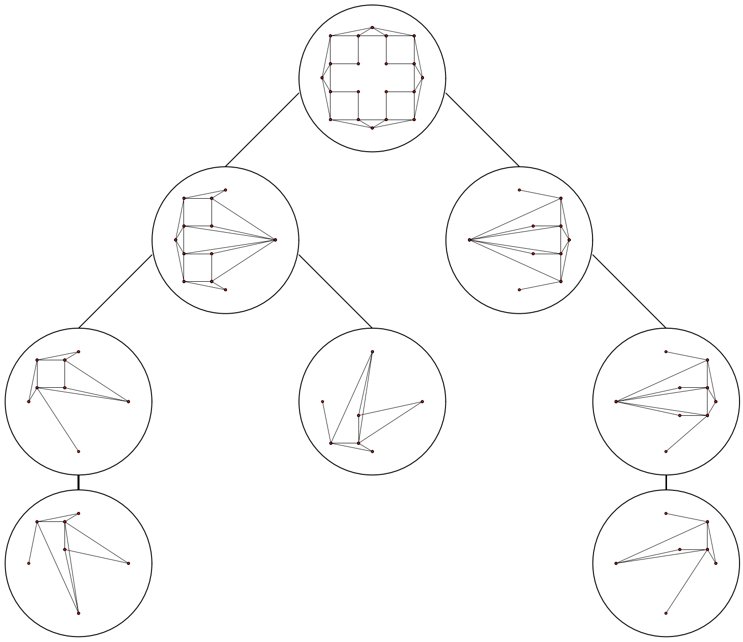
Thus, applying the basic step to all nodes of the tree, we have the total running time of $\mathcal{A}_2$ at most $O(n^2 \log n)$, as required. □

It is noticeable that Algorithm $\mathcal{A}_2$, besides estimation of the treewidth, approximates another parameter of the planar graph, namely the side size of the largest grid minor. It is well known that planar graphs having a large treewidth must have also a large grid as a minor; see e.g. [17, 19, 36]. The following theorem and corollary present the algorithmic consequences of this fact.

Theorem 3.2 *For any planar graph G, the lower bound on the treewidth returned by Algorithm $\mathcal{A}_2$ satisfies the inequality $b/4 \leq LB_2 \leq b$, where b is a side size of the largest square grid minor in G.*



(a) graph



(b) rooted tree

Figure 2: Graph and rooted tree of Algorithm $\mathcal{A}_2$ (trivial child nodes are left out)

Proof: Let M be the largest square grid minor in G with side length b. By construction, LB_2 is the side size of a square grid minor of G. Therefore, inequality $LB_2 \leq b$ always holds. Thus, it remains to prove the lower bound for LB_2.

Consider a node i in the search tree such that G_i contains at least $b^2/2$ vertices of M and each of its children contains at most $b^2/2$ vertices of M. Notice that such a node does always exist otherwise we have a contradiction to the stoppage criteria of the algorithm.

In each node of the search tree, the graphs associated with its child nodes are some connected components in $G \setminus S$ where S consists of at most four vertex cuts obtained in the parent nodes (at most two vertex cuts determining $(\mathcal{N}, \mathcal{S})$ connectivity and at most two vertex cuts determining $(\mathcal{E}, \mathcal{W})$ connectivity). Let S_i be the union of these vertex cuts determining the child nodes of i, i.e. the child nodes of i are some connected components in $G \setminus S_i$. By definition of node i, we have that $G \setminus (V_i \cup S_i)$ contains at most $b^2/2$ vertices of M and each of the child nodes of i contains also at most $b^2/2$ vertices of M.

For a graph $G' = (V', E')$, let us refer to a subset $S \subseteq V'$ as to *balanced separator* of G' if each connected component in $G' \setminus S$ has at most $|V'|/2$ vertices. It is well known that a minimum balanced separator in a $b \times b$ grid graph has size b (see e.g., [18]; the size of a minimum balanced separator is also known as the vertex bisection of the graph.) It is well known, see e.g., [18], that a minimum balanced separator in $b \times b$ grid graph has size b. From observation above we have that $S_i \cap M$ is a balanced separator in M, and therefore $|S_i| \geq b$. Since S_i consists of at most 4 vertex cuts, one of these cuts must have size at least $b/4$ that completes the proof. $\square$

From Theorem 3.2 and the result by Robertson, Seymour, and Thomas [36] that every planar graph of treewidth $tw(G)$ has an $\Omega(tw(G)) \times \Omega(tw(G))$ grid graph as a minor, we deduce the following corollary.

Corollary 3.3 *Algorithm $\mathcal{A}_2$ is a constant approximation algorithm for the treewidth and for the branchwidth of planar graphs.*

Proof: From Theorem 3.2 we have that $S = \Theta(LB_2)$. From the result by Robertson, Seymour, and Thomas [36], we have that $tw(G) = \Theta(bw(G)) = \Theta(S) = \Theta(LB_2)$, where $bw(G)$ is the branchwidth of graph G. $\square$

We complete this section with a brief discussion of advantages and disadvantages of Algorithm $\mathcal{A}_2$ in comparison with the known approximation algorithms for the treewidth on planar graphs. Seymour and Thomas have shown that the branchwidth of a planar graph can be computed in polynomial time [38]. By the result of Robertson and Seymour [34] that each branch decomposition can be converted to a tree decomposition with width at most 3/2 times the width of the branch decomposition (cf. Section 1), we directly have a polynomial time approximation algorithm for treewidth of planar graphs with ratio 3/2.

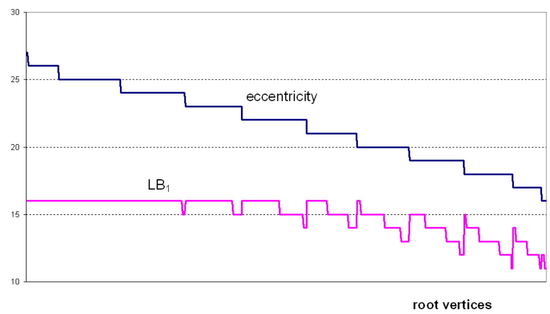
The decision version of the algorithm of Seymour and Thomas [38] runs in $O(n^2)$ time. A version that also constructs branch decompositions of optimal width uses more time; recently, Gu and Tamaki [23] showed this can be done in $O(n^3)$ time. If we want to obtain lower bounds on the treewidth, we only need to run the decision version. However, an important disadvantage of the Seymour and Thomas algorithm is that it is not memory friendly and even for the medium size graphs it easily runs out of memory; see [24, 25]. Recently, Hicks in [25] made several attempts to get a memory friendly algorithm based on the Seymour and Thomas ideas. He derived two memory friendly algorithms with running time $O(n^3)$. Clearly, the machinery of Algorithm $\mathcal{A}_2$ does not require much of memory resources and the algorithm has nearly the same running time as Seymour and Thomas' algorithm, and better running time than Hicks' algorithm. The disadvantage of Algorithm $\mathcal{A}_2$ is its worse performance ratio in comparison to the other algorithms. On the other hand, Algorithm $\mathcal{A}_2$ eventually estimates another important parameter of a planar graph, namely, the side size of the largest square grid minor. Moreover, the version of Algorithm $\mathcal{A}_2$ that also constructs a square grid minor with side size at least 1/4 of the side size of the largest grid minor, runs in the same $O(n^2 \log n)$ time.

4 Computational experiments

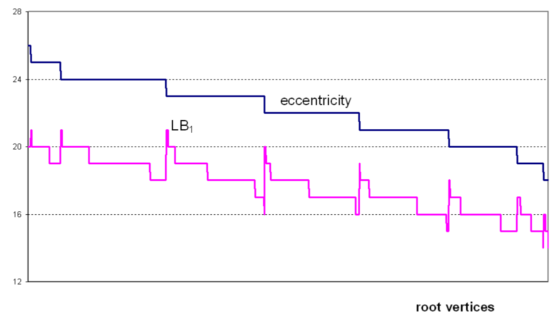
In order to compare in practice the quality of Algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ with previously studied treewidth lower bounds, these algorithms have been implemented in C++ and tried out on a collection of input graphs. In this section, we report on the obtained results for two selected sets of instances. The first set contains a number of general graphs, that have been used in previous studies [11, 28] and originate from different applications like probabilistic networks, frequency assignment, and vertex coloring. The second set of instances consists of planar graphs that have been used by Hicks [24, 25] before. From both sets we selected some instances that are representative for the whole set and/or show an interesting behavior. The CPU times reported are in seconds and obtained on a Linux-operated PC with 3.0 GHz Intel Pentium 4 processor. Some extensive experimental results can be found on [39].

Algorithm $\mathcal{A}_1$ for general graphs has been tested on both the selected planar and non-planar graphs. In the implementation, we skip a graph $G_{i,j}$, whenever $j-i+1$ is not larger than the current value of *bestlow*, as such graphs cannot help to increase the lower bound. This speeds up the algorithm considerably.

As pointed out before, the maximum can be increased further by taking all vertices as root vertex r once. The additional $O(n)$ complexity of the algorithm can be reduced in practice by sorting or limiting the number of root vertices. In principle the algorithm has to be executed for only one of the vertices for which the maximum is achieved. However, we cannot select on this value before computing it. Experiments have shown that the *eccentricity* of a vertex is a reasonable criterion to sort/limit the root vertices. The eccentricity $\varepsilon(v)$ of a vertex v is the maximum depth of a breadth first search with v as root, cf. [40] and hence the best lower bound with root r is limited by $\varepsilon(r)$. Figure 3 shows for two planar graphs the eccentricity and LB_1 for all possible root vertices, sorted first according to non-increasing eccentricity and second to non-increasing LB_1. If the best bound achieved so far is at least $\varepsilon(r)$ for some $r \in V$, we do not have to run algorithm $\mathcal{A}_1$ with r as root. By sorting the vertices according to non-increasing eccentricity, the number of root vertices for which the algorithm should be executed is limited in our computations this way. Figure 3 in fact indicates that the computation times can be reduced further by limiting the number of root vertices to those with high eccentricity, since it is very unlikely that vertices with $\varepsilon(v)$ not close to the maximum achieve the maximum LB_1.



(a) u724



(b) nrw1379

Figure 3: LB_1 and eccentricity for all possible root vertices

Table 1 reports the results for non-planar graphs, in comparison with a contraction degeneracy $\delta C(G)$ lower bound [11]. Different behavior can be observed: for the graphs originating from probabilistic networks, frequency assignment, and coloring, LB_1 is outperformed by the contraction degeneracy, both in time and value. For the graphs originating from a solution approach for the traveling salesman problem, LB_1 is significantly higher than $\delta C(G)$. It is known that these graphs are *close to* planar, which restricts the contraction degeneracy to exceed small values (i.e., $\delta C(G) \leq 5 + \gamma(G)$, where $\gamma(G)$ is the genus of G [41]). As was hoped for, the new lower bound turns out to be profitable for exactly those instances, closing the gap to the best known upper bound UB substantially. Table 2 shows the results for planar graphs. The same behavior as for the close to planar graphs can be observed.

instance	$\lvert V \rvert$	$\lvert E \rvert$	δC	CPU	LB_1	CPU	UB
link	724	1738	11	0.02	7	89.95	13
munin1	189	366	10	0.00	4	2.19	11
munin3	1044	1745	7	0.01	4	195.35	7
pignet2	3032	7264	38	0.12	5	3456.77	135
celar06	100	350	11	0.00	3	1.50	11
celar07pp	162	764	15	0.01	3	19.40	18
graph04	200	734	20	0.02	5	0.38	55
school1	385	19095	122	0.59	3	46.22	188
school1-nsh	352	14612	106	0.39	3	45.34	162
zeroin.i.1	126	4100	50	0.04	2	0.30	50
fl3795-pp	1433	3098	6	0.04	6	1501.53	13
fnl4461-pp	1528	3114	5	0.05	14	4703.67	33
pcb3038-pp	948	1920	5	0.03	12	383.77	25
rl5915-pp	863	1730	5	0.02	10	470.76	23
rl5934-pp	904	1800	5	0.02	12	378.17	23

Table 1: Results for algorithm $\mathcal{A}_1$ on selected non-planar graphs. The last five instances originate from [16] and close to planar. They have been pre-processed by the rules in [10] to reduce the size of the graphs.

Table 2 also shows the lower bound LB_2 computed by algorithm $\mathcal{A}_2$. As initial outer face, the longest face of the computed planar embedding is taken, and partitioned in (roughly) equally sized parts $North$, $East$, $South$, and $West$.

Comparing LB_1 and LB_2, there is no clear winner. In some cases LB_2 is better than LB_1, but more often it is slightly worse. The computation time of LB_2 is however significantly less than that of LB_1, which could be of importance if the bound is incorporated in a branch and bound approach.

The 3/2-approximation algorithm of Seymour and Thomas [38] for planar graphs computes in fact the branchwidth $\beta(G)$. It is well-known that the $\beta(G) - 1$ is a lower bound on the treewidth. The values reported in Table 2 are taken from Hicks [24]. In all cases, this lower bound is higher than LB_1 and LB_2, as is the computational effort, cf. [24].

instance	$\|V\|$	$\|E\|$	δC	CPU	LB_1	CPU	LB_2	CPU	$\beta(G)-1$
d1655	1655	4890	5	0.04	20	995.62	17	13.76	28
d2103	2103	6290	5	0.07	23	1331.97	24	22.21	28
nrw1379	1379	4115	5	0.04	21	567.85	20	9.56	30
pr1002	1002	2972	5	0.03	16	605.80	17	4.69	20
pr2392	2392	7125	5	0.07	21	16391.82	19	21.46	28
tsp225	225	622	5	0.01	10	15.68	9	0.43	11
u2152	2152	6312	5	0.06	23	60192.11	23	12.53	30
u2319	2319	6869	5	0.06	41	2625.04	31	25.64	43
u724	724	2117	5	0.02	16	550.48	14	2.98	17

Table 2: Results for algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ on selected planar graphs.

5 Concluding remarks

The treewidth of a graph can be characterized by the notion of brambles, introduced by Seymour and Thomas [38]. In this work, we developed bramble construction algorithms as to bound the treewidth from below. The constructed brambles turn out to be profitable, both in theory and practice, for graphs that are (*close to*) planar. These results complement previously studied treewidth lower bounds that turned out to be good for graphs that are *far from* planar.

The brambles searched for in this paper have structures that allow easy construction and order computation. An interesting question for further research is whether more such structures exist. These may perhaps lead to new lower bound algorithms for treewidth. Experiments (see e.g., [39]) reveal that for many instances, taken from applications, there still are significant gaps between the best upper and lower bounds, especially for graphs that are (close to) planar; thus, while this paper makes a usefull step, the quest for better algorithms for determining or approximating the treewidth of such graphs remains.

Acknowledgements

We thank Illya Hicks for providing us with the planar graphs for our experiments.

References

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

[2] E. Amir. Efficient approximations for triangulation of minimum treewidth. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 7–15, 2001.

[3] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[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 J. Comput.*, 25:1305–1317, 1996.

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

[7] H. L. Bodlaender. Discovering treewidth. In P. Vojtáš, M. Bieliková, and B. Charron-Bost, editors, *SOFSEM 2005: Theory and Practive of Computer Science: 31st Conference on Current Trends in Theory and Practive of Computer Science*, pages 1–16. Springer-Verlag, Lecture Notes in Computer Science 3381, 2005.

[8] H. L. Bodlaender and A. M. C. A. Koster. On the Maximum Cardinality Search lower bound for treewidth. In J. Hromkovič, M. Nagl, and B. Westfechtel, editors, *Proc. 30th International Workshop on Graph-Theoretic Concepts in Computer Science WG 2004*, pages 81–92. Springer-Verlag, Lecture Notes in Computer Science 3353, 2004.

[9] H. L. Bodlaender and A. M. C. A. Koster. Safe separators for treewidth. In *Proceedings 6th Workshop on Algorithm Engineering and Experiments ALENEX04*, pages 70–78, 2004.

[10] 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.

[11] H. L. Bodlaender, A. M. C. A. Koster, and T. Wolle. Contraction and treewidth lower bounds. In S. Albers and T. Radzik, editors, *Proceedings 12th Annual European Symposium on Algorithms, ESA2004*, pages 628–639. Springer, Lecture Notes in Computer Science, vol. 3221, 2004.

[12] H. L. Bodlaender, A. M. C. A. Koster, and T. Wolle. Contraction and treewidth lower bounds. *Journal of Graph Algorithms and Applications*, forthcoming.

[13] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31:212–232, 2001.

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

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

[16] W. Cook and P. D. Seymour. Tour merging via branch-decomposition. *INFORMS J. on Computing*, 15(3):233–248, 2003.

[17] E. D. Demaine and M. Hajiaghayi. Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA2005*, pages 682–689, 2005.

[18] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34:313–356, 2002.

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

[20] F. V. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, pages 568–580, 2004.

[21] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

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

[23] Q.-P. Gu and H. Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming.* Lecture Notes in Computer Science, Springer, 2005.

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

[25] I. V. Hicks. Planar branch decompositions II: The cycle method. *INFORMS J. on Computing*, 17:413–421, 2005.

[26] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21:549–568, 1974.

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

[28] 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.

[29] B. Lucena. A new lower bound for tree-width using maximum cardinality search. *SIAM J. Disc. Math.*, 16:345–353, 2003.

[30] K. Menger. Zur allgemeinen Kurventheorie. *Fund. Math.*, 10:96–115, 1927.

[31] S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM J. Disc. Math.*, 10:146–157, 1997.

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

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

[34] N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory Series B*, 52:153–190, 1991.

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

[36] N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Comb. Theory Series B*, 62:323–348, 1994.

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

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

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

[40] D. B. West. *Introduction to graph theory.* Prentice Hall, 2001.

[41] 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.