New Upper Bound Heuristics for Treewidth

Emgad H. Bachoore

Hans L. Bodlaender

institute of information and computing sciences, utrecht university
technical report UU-CS-2004-036
www.cs.uu.nl

New Upper Bound Heuristics for Treewidth[*]

Emgad H. Bachoore Hans L. Bodlaender
Institute of Information and Computing Sciences,
Utrecht University, P.O. Box 80.089, 3508 TB Utrecht,
The Netherlands

Abstract

In this paper, we introduce and evaluate some heuristics to find an upper bound on the treewidth of a given graph. Each of the heuristics selects the vertices of the graph one by one, building an elimination list. The heuristics differ in the criteria used for selecting vertices. These criteria depend on the fill-in of a vertex and the related new notion of the fill-in-excluding-one-neighbor. In several cases, the new heuristics improve the bounds obtained by existing heuristics.

1 Introduction

For several applications involving graphs, it is of great interest to have good algorithms that compute or approximate the treewidth of a graph and its corresponding tree decomposition. The interest in these notions, and some other related notions such as branchwidth, branch decomposition, pathwidth, path decomposition, and minimum fill-in arose because of their theoretical significance in (algorithmic) graph theory, and because a tree decomposition of small width of a graph enables us to solve many graph problems in linear or polynomial time. Among these problems are many well-known combinatorial optimization problems such as: graph coloring [17], maximum independent set, and the Hamiltonian cycle problem. Nowadays, there are several 'real world' applications that use the notion of tree decomposition or branch decomposition to find solutions for the problems at hand. These come from many different fields, such as expert systems [14], probabilistic networks [3], frequency assignment problems [11, 12], telecommunication networks design, VLSI-design, natural language processing [9], and the traveling salesman problem [8].

The problem of computing the treewidth of a graph is *NP-hard* [1]. Therefore, for computing the treewidth of a graph, we have to use an exact but slow method like branch and bound, algorithms that work only for specific classes of graphs, or resort to algorithms that only approximate the treewidth. In the past years, several heuristics for treewidth have been designed. We can divide those heuristics into two categories: those that find upper bounds for the treewidth, and those that find lower bounds for the treewidth. This paper concentrates on upper bound heuristics.

[*] This work has been supported by the Netherlands Organization for Scientific Research NWO (project TACO: 'Treewidth And Combinatorial Optimization').

Some of the heuristics for finding an upper bound for the treewidth are based on algorithms that test whether a given graph is triangulated. These are Maximum Cardinality Search, Lexicographic Breadth First search, Minimum Degree and Minimum Fill-in algorithms. Other heuristics are based on other ideas, e.g. the Minimum Separating Vertex Set algorithm (cf. [6, 10]).

In this paper, we present a number of heuristic methods to find upper bounds for the treewidth of the graph. Each of our heuristics is based on constructing a triangulation of the graph from an elimination ordering of the vertices. These elimination orderings are constructed by repeatedly selecting a vertex and then adding the so-called fill-in edges between its neighbors. The various heuristics differ in the criteria of the selection of the vertices. These criteria, basically, depend on two concepts: the number of edges that must be added between the neighbor vertices of a vertex to form a clique, and the number of edges that must be added between all neighbor vertices of a vertex except one to form a clique between them.

We have implemented the algorithms proposed in this paper, and tested them on a set of 32 graphs. These graphs are taken from instances of probabilistic networks, frequency assignment and vertex coloring. We compared the results of these algorithms with those of other heuristic methods. We observed that in many cases our algorithms perform well.

2 Definitions and preliminary results

In this section, we give the definitions of the most frequently used concepts and notions in this paper. Let $G= (V,E)$ be an ***undirected graph*** with vertex set V and edge set E. A graph H is a ***minor*** of graph G, if H can be obtained from G by zero or more vertex deletions, edge deletions, and edge contractions. ***Edge contraction*** is the operation that replaces two adjacent vertices v and w by a single vertex that is connected to all neighbors of v and w.

We denote the set of neighbors of vertex v by $\boldsymbol{N(v)= \{w \in V \mid \{v, w\} \in E\}}$, and the set of neighbors of v plus v itself by $\boldsymbol{N[v]= N(v) \cup \{v\}}$. In the same manner we define $\boldsymbol{N^0[v]= \{v\}}$, $\boldsymbol{N^{i+1}[v]= N[N^i[v]]}$, $\boldsymbol{N^{i+1}(v)=N^{i+1}[v] \backslash N^i[v]}$. We can extend the above definition to a set of vertices instead of one vertex. Suppose that S is a set of vertices, then $\boldsymbol{N^0[S]= S}$, $\boldsymbol{N^{i+1}[S]= N[N^i[S]]}$, $\boldsymbol{N^{i+1}(S)= N^{i+1}[S] \backslash N^i[S]}$, $\boldsymbol{N[S]= \bigcup_{v\in S} N[v]}$, $\boldsymbol{N(S)= N[S] \backslash S}$, $\mathbf{i} \in \mathbb{N}$.

Let $\boldsymbol{degree(v)= |N(v)|}$ be the degree of vertex v. A graph is ***complete*** if every pair of its distinct vertices is adjacent. Given a subset $A \subseteq V$ of the vertices, we define the ***subgraph induced* by *A*** to be $G_A= (A, E_A)$, where $E_A = \{\{x, y\} \in E: x \in A \text{ and } y \in A\}$. A subset $A \subseteq V$ of r vertices is an ***r-clique*** if it induces a complete subgraph. A single vertex is a 1-*clique*. A subset $A \subseteq V$ is a clique if it is an r-clique for some r. A clique A is ***maximal***, if G has no clique B that contains A as a proper subset. A clique is ***maximum*** if there is no clique of G of larger cardinality. The ***clique number of G***, $\boldsymbol{\omega(G)}$ is the number of vertices in a maximum clique of G; i.e., it is the size of the largest complete subgraph of G. A subset $A \subseteq V$ of r vertices is an ***r-almost-clique*** if there is a $v \in A$ such that $A - \{v\}$ forms a clique. A vertex v in G is called ***simplicial***, if its set of neighbors $N(v)$ forms a clique in

G. A vertex v in G is called ***almost simplicial***, if its neighbors except one form a clique in G, i.e., if v has a neighbor w such that $N(v) - \{w\}$ is a clique. A graph G is called ***triangulated*** if every cycle of length four of more possesses a *chord*. A ***chord*** is an edge between two nonconsecutive vertices of the cycle. Triangulated graphs are also called ***chordal graphs***. A graph $G=(V, E)$ is a ***subgraph*** of graph $H=(W, F)$ if $V \subseteq W$ and $E \subseteq F$. A graph $H= (W, F)$ is ***a triangulation*** of graph $G=(V, E)$, if G is a subgraph of H and H is a triangulated graph.

A ***linear ordering*** of a graph $G= (V, E)$ is a bijection f: $V \rightarrow \{1, 2, \ldots, |V|\}$. A linear ordering of the vertices of a graph G, $\sigma = [v_1, \ldots, v_n]$ is called a ***perfect elimination order*** (***p.e.o.***) of G, if for every $1 \leq i \leq n$, v_i is a simplicial vertex in $G[\{v_i, \ldots, v_n\}]$, i.e., the higher numbered neighbors of v_i form a clique. It has been shown in [7] that a graph G is triangulated, if and only if G has a *p.e.o*. ***Eliminating a vertex v*** from a graph $G = (V, E)$ is the operation that first adds an edge between every pair of non-adjacent neighbors of v, and then removes v and its incident edges.

The ***tree decomposition*** of the graph $G = (V, E)$ is a pair (X, T) in which $T = (I, F)$ a tree, and $X= \{X_i \mid i \in I\}$ a collection of subsets of V, one for each node of T, such that:

- $\bigcup_{i \in I} X_i = V$,
- For all $(u, v) \in E$, there exists an $i \in I$ with $u, v \in X_i$, and
- For all $i, j, k \in I$: if j is on path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

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

Lemma 1. (See Bodlaender [4].)

(i) For every triangulated graph $G = (V, E)$, there exists a tree decomposition $(X= \{X_i \mid i \in I\}, T= (I, F))$ of G, such that every set X_i forms a clique in G, and for every maximal clique $W \subseteq V$, there exists an $i \in I$ with $W= X_i$.

(ii) Let $(X= \{X_i \mid i \in I\}, T= (I, F))$ be a tree decomposition of G of width at most k. The graph $H= (V, E \cup E')$, with $E'= \{\{v, w\} \mid \exists\, i \in I : v, w \in X_i\}$, obtained by making every set X_i a clique, is triangulated, and has maximum clique size at most $k + 1$.

(iii) Let $(X= \{X_i \mid i \in I\}, T= (I, F))$ be a tree decomposition of G, and let $W \subseteq V$ form a clique in G. Then there exist an $i \in I$ with $W \subseteq X_i$.

Lemma 2. (See Shoikhet and Geiger [16].) For triangulated graphs, tree decompositions exist where the nodes are exactly the maximal cliques of the graph. Such tree decompositions are called ***clique trees***. The ***width of a triangulated graph T*** is $\max_{k \in K(T)} (|K| - 1)$, where $K(T)$ is the set of *maximal cliques* of T.

3 Upper Bound Heuristics for Treewidth

In this section we present some heuristic methods to find upper bounds for the treewidth of a given graph, and the corresponding tree decompositions. Basically, these methods depend on two concepts: The first one is the number of edges that must be added between the neighbors of a vertex x to make it *simplicial*, i.e., the neighborhood of that vertex turn into a *clique*. We call this the *fill-in* of x.

$$\textbf{\textit{fill-in}}(x) = |\{\{v, w\} \mid v, w \in N(x),\ \{v, w\} \notin E\}|$$

The second concept is very similar to the first one, but here we find the minimum number of edges which when added between pairs of neighbors of a vertex x, turn x into an almost simplicial vertex, i.e., by adding these edges to the graph, the neighborhood of that vertex will turn into an almost clique. We call this parameter the fill-in of x excluding one neighbor "*fill-in-excl-one*(x)"

$$\textbf{\textit{fill-in-excl-one}}(x) = \min_{z \in N(x)} |\{\{v, w\} \mid v, w \in N(x) - \{z\}, \{v, w\} \notin E\}|$$

A graph with $|V|$ vertices has $|V|!$ (permutations of $|V|$) linear ordering. For each linear ordering σ of G, we can build a triangulation H_σ of G, such that σ is the *p.e.o.* of $H_{\sigma,}$ in the following way. For $i = 1 \ldots |V|$, in that order, we add an edge between every pair of non-adjacent neighbors of v_i that are after v_i in the ordering, v_i is the i'th vertex in σ. One can observe that σ is a *p.e.o.* of the resulting graph H_σ. As H_σ is triangulated, its treewidth is one smaller than its maximum clique size, which equals the maximum number of neighbors of v over all vertices v that are after v in the linear ordering σ. One can construct from H_σ a tree decomposition of H_σ and of G with width exactly this maximum clique size minus one. It is also known that there is at least one linear ordering of G where we obtain the exact treewidth of G in this way [4]. This suggests the following general scheme for heuristics for treewidth.

```
G' = G;
i = 1; σ = ( );
while G' is not the empty graph
do
    select according to some criteria a vertex v from G';
    eliminate v;
    add v to position i in the ordering σ;
    i = i + 1;
enddo
{Now σ is a linear ordering of V.}
Construct triangulation H_σ of G and the corresponding tree decomposition.
```

Instead of constructing H_σ after σ is constructed, we also can construct H_σ and the corresponding tree decomposition while σ is constructed.

The width of the tree decomposition thus obtained is the maximum over all vertices *v* of the number of neighbors of *v* in *G*'. We call a graph *G*' encountered during the algorithm a *temporary* graph. A linear ordering of *G*, used in this way, is called often an *elimination scheme*. Several heuristics are of this type. Most known are the Minimum Fill-in heuristic, and the Minimum Degree heuristic. In these, we repeatedly select the vertex *v* with minimum fill-in in *G*', or minimum degree in *G*' respectively. These two heuristics appear to be successful heuristics for treewidth; they often give good bounds and are fast to compute. The success of these heuristics encouraged us to develop other heuristics based on similar principles. Our heuristics are inspired by results on preprocessing graphs for treewidth. In [3], reduction rules are given that are safe for treewidth. Here, such rules rewrite a graph *G* to a smaller graph *G'*, and possibly update a variable *low* that gives a lower bound on the treewidth of the original graph.

In [3], the notion of ***safe rule*** was introduced. The safe rule rewrites a graph to a smaller one, and maintains a lower bound variable *low*, such that the maximum of *low* and the treewidth of the graph at hand stays invariant, i.e., rule *R* is safe, if for all graphs *G*, *G'*, and all integers *low*, *low'*, we have

$$\mathbf{(\mathit{G}, \mathit{low}) \rightarrow_{\mathit{R}} (\mathit{G'}, \mathit{low'}) \Rightarrow \max(\mathit{treewidth}(\mathit{G}), \mathit{low}) = \mathit{max}(\mathit{treewidth}(\mathit{G'}), \mathit{low'}).}$$

Thus the treewidth of the original graph is known when we know the treewidth of the reduced graph and *low*. Amongst others, the following two rules were shown to be safe for the treewidth in [3].

The Simplicial Reduction Rule (SRR):
Let v be a simplicial vertex of degree(v) ≥ 0.
Remove v.
Set low to max(*low*, *degree*(*v*)).

The Almost Simplicial Reduction Rule (ASRR):
Let v be an almost simplicial vertex of degree(v) ≥ 2.
If low $\geq$ *degree*(v) *then eliminate v.*

The *safeness* of the simplicial reduction rule tells us that if a vertex *v* has *fill-in* zero, then selecting that vertex as the next one in the elimination ordering will not cause a treewidth that is larger than necessary for this elimination ordering. It can be seen as a motivation for the Minimum Fill-in Heuristic, where we select vertices with minimum fill-in. Similarly, the almost simplicial reduction rule can be seen as motivation to look at the *fill-in-excl-one*. If a vertex has fill-in-excl-one of zero, then selecting that vertex as the next vertex in the elimination ordering will in many cases not cause the treewidth caused by the formed elimination ordering to be larger than necessary, unless the degree of the almost simplicial vertex is more than the treewidth of the original graph. With a small twist to the terminology, we say that *eliminating v is safe* (in a graph *G*), if the choice of *v* in the heuristic scheme presented above can lead to a tree decomposition whose width equals the treewidth of *G*.

Motivated by these observations, we designed new heuristics for treewidth that are given below.

3-1 Method 1: Enhanced Minimum Fill-in (EMF):

The motivation for the Enhanced Minimum Fill-in algorithm is based upon the following lemmas:

Lemma 3.
1- If v is a simplicial vertex, then eliminating v is safe.
2- If v is an almost simplicial vertex and the treewidth of G is at least the degree of v, then eliminating v is safe.

Proof. See [3]. The lemma is a reformulation of the safeness of the Simplicial Reduction Rule and the Almost Simplicial Reduction Rule.

Note that when we have vertices x and y with *fill-in-excl-one*(y)= 0, degree(y) $\leq$ *low* (for some lower bound *low* on the treewidth of the input graph), *fill-in*(x) $>$ 1, and *fill-in*(y) > *fill-in*(x), then y appears to be the best choice for elimination (as this is safe by lemma 3); the Minimum Fill-in heuristic, however, would have selected x.

For faster implementation of the algorithms, we observe that in many cases we do not have to recompute the values of *fill-in* and *fill-in-excl-one* of every vertex in the temporary graph after we eliminate a vertex from it.

Lemma 4. Let v be a simplicial vertex in graph G, $G' = G[V\text{-}\{v\}]$ be the graph obtained by eliminating v, and *fill-in*$_G(v)$ be the *fill-in* of vertex v in graph G. For all $w \notin N^1(v)$, we have

$\quad$ *fill-in*$_G(w)$= *fill-in*$_{G'}(w)$,
$\quad$ *fill-in-excl-one*$_G(w)$= *fill-in-excl-one*$_{G'}(w)$.

Therefore, when we eliminate a simplicial vertex v from a graph, and we want to find the next vertex in the graph with minimum fill-in or minimum fill-in-excl-one, we need only to recompute *fill-in* and *fill-in-excl-one* for neighbors of v ($N^1(v)$) . For instance, if we eliminate vertex 1 from the graph in Figure 1 and we want to find the next vertex with minimum fill-in or minimum fill-in-excl-one in the graph, then we need only to recompute the *fill-in* and *fill-in-excl-one* for vertices 2, 3 and 4.

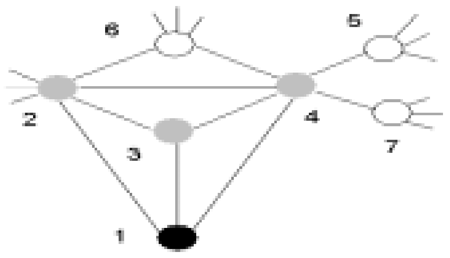

Figure 1.

Similarly, the following lemma shows that if we eliminate a non simplicial vertex v from a graph, and we want to find the next vertex in the graph with minimum fill-in or minimum fill-in-excl-one, then the *fill-in* and *fill-in-excl-one* of only the vertices in $N^2(v)$ could be change. For instance, if we eliminate vertex 1 from the graph in Figure 2 and we want to find the next vertex with minimum fill-in or minimum fill-in-excl-one, then we need to recompute the *fill -in* and *fill-in-excl-one* for vertices 2, 3, 4 and 5 only.

Lemma 5: Let v be any vertex in graph G that is not simplicial, $G' = G[V\text{-}\{v\}]$ be the graph obtained by eliminating v, and $\mathit{fill\text{-}in}_G(v)$ be the *fill-in* of vertex v in graph G. For all $w \notin N^2(v)$, we have

$$\mathit{fill\text{-}in}_G(w) = \mathit{fill\text{-}in}_{G'}(w),$$
$$\mathit{fill\text{-}in\text{-}excl\text{-}one}_G(w) = \mathit{fill\text{-}in\text{-}excl\text{-}one}_{G'}(w).$$

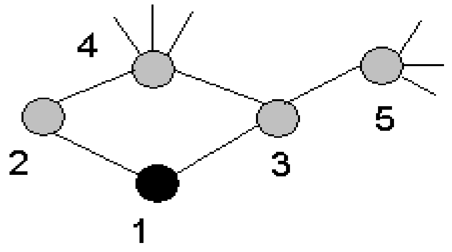

Figure 2.

Motivated by these observations, we propose the Enhanced Minimum Fill-in algorithm. Here, we first select vertices whose elimination is safe by Lemma 3, i.e., we select vertices that are simplicial (have *fill-in* 0), or almost simplicial (have *fill-in-excl-one* 0) and their degrees are at most the lower bound (*low*) on the treewidth of the graph. Otherwise, we select the vertex of *minimum fill-in*. See Algorithm (1) for more details.

Observation

Many of the heuristics have slightly different implementations that can give different results on the same graph. For instance, consider the Minimum Fill-in heuristic. If there is more than one vertex that has minimum fill-in, the method does not specify which of these has to be selected and placed in the elimination ordering. For instance, there could be some arbitrary numbering of the vertices, and the specific implementation could choose the lowest or the highest numbered vertex with minimum fill-in. It seems better to use criteria that guide towards a better upper bound for the treewidth for such a selection. Using other criteria apart from minimum fill-in can be seen to give better bounds for several inputs. Still, in each case, there are graphs for which we do not find the optimal treewidth with such heuristics; given the *NP-hardness* of the problem, we also cannot expect to do so.

We would like to remark here that if we want to fully describe an upper bound algorithm, we must specify details like the method of representing the graph, specifically, because of matters like when vertices have the same fill-in, in which manner such a tie is broken. If we do not give such specifications, there is possibility to get different results from the same algorithm. Moreover, it becomes more difficult to compare the results of different methods.

```
Algorithm 1: Enhanced Minimum Fill-in (EMF):
Input: A connected graph G(V,E), and a lower bound on the treewidth of G, low.
Output: An upper bound on the treewidth of the input graph G, ub(G) and its corresponding tree
        decomposition, td(G) , and elimination order, σ(G).

/* phase 1: Initialization */
let H(W, F)= G(V, E); ub(H)= 0;  td(H)= ∅;  σ(G) = ∅;

for every $w \in W$
    compute degree[w];
    compute fill-in[w];
    compute fill-in-excl-one[w];
 /* phase 2: the main phase */
while H is not empty /*determine which vertex should be eliminated from the graph.*/
do
   if (H has a simplicial vertex w)
   then update(H, w);
   else if  (H has an almost simplicial vertex w with degree(w) $\leq$ low)
          then
              update(H, w);
          else
              find the vertex w with minimum fill-in in H;
              update(H, w);
enddo;
return(ub(G) , td(G), σ(G));
End Algorithm 1: Enhanced Minimum Fill-in.
```

```
Algorithm 2: update(G, p)  /* eliminate a vertex v from a graph G, and update the values of the upper
bound for treewidth, and the corresponding  tree decomposition and elimination order;  and recompute
the fill-in, fill-in-excl-one and the degrees for some vertices, see Lemmas 4 and 5  */
ub(G) = max(ub(G) , degree(p));
add p to the end of the σ;
create a new node (bag) $X_i$ in td(G), let $X_i$= N[p];  /*add p with its neighbors as a node to the td(G) */
/* recompute the fill-in and fill-in-excl-one for some vertices in H, see lemmas 3 & 4 */
if (p is simplicial)
then
    for every $x \in N^1(p)$
        compute fill-in(x);,
        compute fill-in-excl-one(x);
else
    clique(p);  /* add an edge between any two non adjacent neighbors of v */
    for every $x, y \in N^1(p)$  and  $\{x, y\} \notin E$   /* update the degrees of the non adjacent neighbors of p */
        degree(x)= degree(x) + 1;
        degree(y)= degree(y) + 1;
    for every $x \in N^2(p)$
        compute fill-in(x) ;
        compute fill-in-excl-one(x);
 for every $x \in N^1(p)$   /* update the degree of every neighbor of vertex p */
     degree(x)= degree(x) – 1;
End Algorithm 2: Update.
```

3-2 Method 2.1: Minimum Fill-in Excluding One Neighbor (MFEO1) :

Using the ideas behind Method 1 and additional techniques, we now develop a more advanced heuristic. The idea is as follows: first, we test whether the graph contains simplicial vertices, or almost simplicial vertices with degree at most the lower bound for the treewidth of the graph. If we find such vertices, we process them as in Method 1. If the graph does not include such simplicial or almost simplicial vertices anymore, then we go to the second step. In this step, we test whether the graph contains a vertex *v* with *fill-in-excl-one* less than the minimum *fill-in* of the vertices in the temporary graph, such that the degree of *v* is at most the current lower bound for the treewidth. In the case that the graph includes such a vertex, then that vertex will be processed first; otherwise, the vertex with minimum *fill-in* should be processed first. However, if the graph contains more than one vertex with such properties then the vertex with minimum *fill-in-excl-one* amongst these is processed first. But, if still there is more than one vertex that satisfies the last condition, then the vertex with minimum *fill-in* amongst these should be selected first.

In the algorithm we distinguish two phases: *The Initialization Phase* and *The Main Phase*. The main phase consists of a loop. Each loop iteration has two main steps: (a) finding the next vertex in the elimination order, and (b) updating the values of the parameters. The input to the algorithm is a connected undirected graph. The algorithm returns an upper bound on the treewidth of the graph and its corresponding tree decomposition. Algorithm 3 gives more details for this method.

Phase I: *The Initialization Phase*:

In this phase, the upper bound for the treewidth of the graph is initialized with the value zero, the tree decomposition with an empty set, and σ with an empty list.

Phase II: *The Main Phase*:

In each iteration of this phase, the following steps are performed:

Step (a): *Finding the next vertex in the elimination order* (σ):

As mentioned above, the algorithm verifies whether the temporary graph contains simplicial vertices or almost simplicial vertices with degree less than or equal to the lower bound for the treewidth. If so, these are processed by placing them on σ. If there are no such vertices, the algorithm finds a vertex *p* with minimum *fill-in* in the temporary graph and then performs the following test on the other remaining vertices. Let *p* be the vertex with minimum *fill-in*. We test if there exists a vertex *w* with the following properties:

(1) *degree*(*w*) < *low*,
(2) *fill-in-excl-one(w)* < *fill-in(p)* = *minimum fill-in*,

If so, such a vertex *w* is added to the elimination order list. In case there are more vertices that satisfy conditions (1) and (2) above, preference is given to the vertex with minimum fill-in-excl-one. If there is still more than one vertex that can be selected, then, amongst these, preference is given to the vertex with minimum fill-in.

Step (b): ***Updating phase*****:**
This phase consists of the following operations to keep the values of the variables defined in phase (1) up to date:

(1) Updating the *upper bound* value. Set the upper bound for the treewidth of the graph to the maximum of the current one and the degree of the vertex selected from *step (a) above*.
Let v be the vertex selected from step (a).
Let upper bound (G)= *max* (*upper bound*(G), *degree*(v)).
(2) Updating the elimination order list: Add vertex v to the next empty position of σ.
(3) Updating the tree decomposition: Add vertex v and its neighbor vertices as a node to the tree decomposition of the graph,
(4) Eliminating v: turning the set of neighbors of v into a clique and then removing v and its incident edges from the graph.
(5) Recompute the *fill-in* and *fill-in-excl-one* of some vertices in the temporary graph: If the eliminated vertex v is simplicial, then recompute the *fill-in* and *fill-in-excl-one* for all neighbors of vertex v, $N^1(v)$; otherwise recompute the *fill-in* and *fill-in-excl-one* for all neighbors of vertex v and the neighbors of those neighbors, $N^2(v)$.
(6) Update the degree of every neighbor vertex of v, $N^1(v)$.

After the Initialization phase, the algorithm repeats the two main steps (a) and (b) until the temporary graph contains no vertices anymore. At the end of the procedure, the final results are produced: an upper bound on the treewidth, its corresponding tree decomposition, and a perfect elimination ordering of the input graph. For more details see Algorithm 3.

3-3 Method 2.2: Minimum Fill-in Excluding One Neighbor Vertex (MFEO2) :

The difference between this heuristic and the previous one is in the method that is used by the algorithm to find the sequence of vertices in the elimination order list, namely, in Phase II, and more precisely in step (a) of that phase. The sequence of operations (conditions) in this step becomes as follows: If we have more than one candidate vertex satisfying conditions (1) and (2) mentioned in section 3-2, then the vertex with minimum *fill-in* amongst these will be selected**.** But, if there still is more than one vertex fulfilling all conditions, then from these, the vertex with minimum *fill-in-excl-one* of those vertices will be selected. In other words, we switched the order in which the *fill-in* and *fill-in-excl-one* are used to do the tie breaking in the end. For more details see algorithm 4.

Algorithm 3: Minimum Fill-in Excluding One Neighbor (MFEO1)

Input: A connected graph ***G(V,E),*** and a lower bound on the treewidth of *G*, ***low.***
Output: An upper bound on the treewidth of the input graph *G*, ***ub*(*G*)** and its corresponding tree decomposition, ***td*(*G*)** , and elimination order, **σ(*G*).**

```
/* phase 1: Initialization */
let $H(W, F) = G(V, E)$; $ub(H) = 0$;  $td(H) = \varnothing$;  $\sigma(G). = \varnothing$;

for every $w \in W$
     compute $degree[w]$;
     compute $fill\text{-}in[w]$;
     compute $fill\text{-}in\text{-}excl\text{-}one[w]$;
 /* phase 2: the main phase */
while $H$ is not empty /*determine which vertex should be eliminated from the graph.*/
do
    if ($H$ has a simplicial vertex $w$)
    then update($H$, $w$)
    else if  ($H$ has an almost simplicial vertex $w$ with $degree[w] \le low$)
          then
                update($H$, $w$);
          else
                find the vertex $p$ that has minimum $fill\text{-}in$ in $H$;
                $minimum\text{-}fill\text{-}in = fill\text{-}in(p)$;
                $local\text{-}min\text{-}fill\text{-}in$= MaxInt;
                $local\text{-}min\text{-}fill\text{-}in\text{-}excl\text{-}one$= MaxInt;

                for all $q \in W$
                do
                   if  (($fill\text{-}in\text{-}excl\text{-}one[q] < min\text{-}fill\text{-}in$) & ($degree[q] \le low$))
                   then

                      if (($fill\text{-}in\text{-}excl\text{-}one[q] < local\text{-}min\text{-}fill\text{-}in\text{-}excl\text{-}one$) or

                         (($fill\text{-}in\text{-}excl\text{-}one[q] = local\text{-}min\text{-}fill\text{-}in\text{-}excl\text{-}one$) and ($fill\text{-}in[q] < local\text{-}min\text{-}fill\text{-}in$)))
                      then
                           $p = q$;
                           $local\text{-}min\text{-}fill\text{-}in = fill\text{-}in[q]$;
                           $local\text{-}min\text{-}fill\text{-}in\text{-}excl\text{-}one = fill\text{-}in\text{-}excl\text{-}one[q]$;
                enddo;
                update($H$, $p$);
enddo;
return($ub(G)$ , $td(G)$, $\sigma(G)$);
End Algorithm 3: Minimum Fill-in Excluding One Neighbor (MFEO1).
```

Algorithm 4: Minimum Fill-in Excluding One Neighbor (MFEO2)

```
/* phase 2: the main phase */
while $H$ is not empty /*determine which vertex should be eliminated from the graph.*/
do
    if ($H$ has a simplicial vertex $w$)
    then update($H$, $w$)
    else if  ($H$ has an almost simplicial vertex $w$ with $degree[w] \le low$)
           then
                update($H$, $w$);
           else
                find the vertex $p$ that has minimum $fill\text{-}in$ in $H$;
                $minimum\text{-} fill\text{-}in = fill\text{-}in(p)$;
```

```
          local-min-fill-in= MaxInt;
          local-min-fill-in-excl-one= MaxInt;
          for all $q \in W$
          do
            if ((fill-in-excl-one[q] < min-fill-in) & (degree[q] $\leq$ low))
            then
               if ((fill-in[q] < local-min-fill-in) or
                  ((fill-in[q] = local-min-fill-in) and (fill-in-excl-one[q] < local-min-fill-in-excl-one)))
               then
                  p= q;
                  local-min-fill-in= fill-in[q];
                  local-min-fill-in-excl-one= fill-in-excl-one[q];
          enddo;
          update(H, p);
enddo;
```

3-4 Method 3.1: The Ratio heuristic, version 1 (Ratio1):

In the two versions of the Ratio heuristic, we use different rules for when vertices of small fill-in-excl-one can be selected for elimination before vertices of minimum fill-in. Again, we first select simplicial vertices, or almost simplicial vertices whose degree is at least the lower bound for the treewidth. If there are no such vertices in the temporary graph, we proceed now as follows: In the Ratio heuristic, version 1, a vertex *v* can be selected when its fill-in-excl-one is smaller than minimum fill-in, its degree is at most the lower bound for the treewidth, and it satisfies the following condition. Let *H*(*W, F*) be a temporary graph of graph *G*(*V*, *E*). Select a vertex *p* of minimum fill-in. Compute $r_1(w)$= *fill-in*(*w*) / *fill-in*(*p*), and $r_2(w)$= *degree*(*w*) / *degree*(*p*). We now require that $r_1(w) < r_2(w)$ for $w \neq p$ to be a candidate for selection at this point. If we have more than one such candidate, we select from these a vertex with minimum difference between *r1* and *r2*, ($r_1 - r_2$).

The motivation for the Ratio heuristic, version 1, is that we want to select vertices for which elimination creates a large clique while only few fill-in edges are added. Let us illustrate the method with the following examples: Let *p* be a vertex of minimum fill-in and *w* be a vertex whose fill-in-excl-one is less than minimum fill-in and its degree is less than the value of the lower bound for the treewidth.

Example 1*:*
Let *minimum fill-in*= *fill-in*(*p*)= 6, *degree*(*p*)= 40, *fill-in-excl-one*(*w*)= 5, *degree*(*w*)= 30, *lower-bound*= 10, $r_1(w)$= 5 / 6, $r_2(w)$= 30/ 40, and $r_1(w)$ is less than $r_2(w)$. In such a case, vertex *w* will be eliminated before vertex *p* from the graph.

Example 2:
Let *minimum fill-in*= *fill-in*(*p*)= 4 *degree*(*p*)= 60, *fill-in-excl-one*(*w*)= 3, *degree*(*w*)= 50, *lower-bound*= 10, $r_1(w)$= 3/ 4, $r_2(w)$= 50/ 60, and $r_1(w)$ is not less than $r_2(w)$. In such a case, vertex *p* will be eliminated before vertex *w* from the graph.

We only show the new part of the algorithm below; phase 1 is as in Algorithm 3.

Algorithm 5: Ratio 1

```
/* phase 2: determine which vertex should be eliminated from the graph.*/
Let H= (W, F) be the temporary graph,
While H is not empty
do
   if (H has a simplicial vertex w)
   then update(H, w)
   else if  (H has an almost simplicial vertex w with degree[w]  ≤  low)
         then
              update(H, w);
         else
              find the vertex p that has minimum fill-in in H;
               let $r_3$ = - ∞;
               for every $w \in W$
               do
                  if ((fill-in-excl-one(w) < min-fill-in) and (degree(w) ≤ low))
                  then
                       $r_1$ = fill-in-exc-one(w) / min-fill-in;
                       r2= degree(w) / degree(p);
                       if (($r_1 < r_2$) and (($r_2 - r_1$)  > $r_3$)))
                       then
                            p= w;
                            $r_3 = r_2 - r_1$;
               enddo;
      update(H, p);
enddo;
```

Algorithm 6: Ratio 2

```
/* phase 2: determine which vertex should be eliminated from the graph.*/
Let H= (W, F) be the temporary graph,
while H is not empty
do
   if (H has a simplicial vertex w)
   then update(H, w)
   else if  (H has an almost simplicial vertex w with degree[w]  ≤  low)
         then
              update(H, w);
         else
              find the vertex p that has minimum fill-in in H;
               minimum-ratio= MaxInt;
              for every $w \in W$
              do
                  r(w)= fill-in(w) / degree(w);
                  if (r(w) < minimum-ratio)
                  then
                       p= w;
                       minimum_ratio= ratio(w);
              enddo;

   update(H, p);
enddo;
```

3-5 Method 3.2: The Ratio heuristic, version 2 (Ratio 2):

The second variant of the Ratio heuristic is similar to the first one, with the following difference. For each vertex $w \in W$, we set $r(w)$= *fill-in*(w) / *degree*(w) (*degree*(w) > 1, otherwise w is simplicial). When there are no simplicial vertices and no almost simplicial vertices with degree at most the treewidth lower bound, we select the vertex w whose ratio r(w) is smallest. Below, we give the code of phase 2 of this method.

4 Computational Analysis

The algorithms described in the previous section and some other algorithms described in [6, 10] have been implemented using Microsoft Visual C++ 6.0 on a Window 2000 PC with a Pentium III 800 MHz processor.

The algorithms were tested on two sets of graphs. The first one includes 18 graphs from real-life probabilistic networks and frequency assignment problems [10]. The second set includes 14 graphs from a DIMACS coloring benchmark [6]. The selected graphs have different number of vertices and edges. Also, the differences between the known upper bounds and lower bounds for the treewidth of many of those graphs are noticeable. As a result of that, it is possible to obtain different results for the upper bound of the same graph by using different heuristics.

In order to be able to achieve good conclusions from this analysis, we analyze our algorithms in different ways. First, we compare the results of the implementations of different methods that have been introduced in this paper on two aspects, namely, the obtained upper bound for the treewidth of the graph and the computer processing time needed by the algorithms to produce the results. The second part of the evaluations compares the heuristics introduced in this paper with known heuristics.

4-1 Comparison between the heuristics introduced in this paper:

In section 3, we have introduced five methods for finding upper bounds for the treewidth of the graph; in addition we implemented the Minimum Fill-in heuristic (MF) and the Minimum Degree Fill-in (MDFI). The results of implementing these algorithms on different graphs are given in the following tables. Table 1(a) and Table 1(b) show a comparison between different methods illustrated in this paper from the point view of their best upper bound values. Table 1(a) consists of instances of probabilistic networks and frequency assignment problems. Table 1(b) gives graphs from the DIMACS coloring benchmark. Table 2(a) and Table 2(b) give the processing times in seconds. These are given for each of the heuristics, and each of the graphs that were used in Table 1(a) and Table 1 (b). Times are rounded to the nearest integer. The LB values are the lower bound values used by the algorithms; these values are not necessarily good or even valid lower bounds for the treewidth.

We notice clearly from the results in these tables that in general the upper bounds obtained by MFEO1 and RATIO2 are better than those obtained by the other methods.

The main differences between the results obtained by MFEO1 and those obtained by RATIO2 are as follows:

- The results of applying the MFEO1 heuristic on graphs of probabilistic networks and frequency assignment instances in Table 1(a) are better than or equal to those produced by any other heuristic in that table. However, this is not the case for some of the instances from the DIMACS coloring benchmark, see Table 1(b). We notice when we consider Table 1(b) that the upper bounds for some graphs are better when using RATIO2 than those obtained when using MFEO1.

- The results of MFEO1 are more stable, always better than or equal to that produced by any other heuristic, except for RATIO2. RATIO2 does not have such a 'stable behavior'.

S	Graph name	|V|	|E|	LB	MF	EMF	MFEO1	MFEO2	RATIO1	RATIO2
1	alarm1	37	65	2	4	4	4	4	4	4
2	barley	48	126	3	7	7	7	7	7	7
3	boblo	221	328	2	3	3	3	3	3	3
4	celar06_pp_a	100	350	11	11	11	11	11	11	11
5	celar07_pp_a	200	817	16	16	16	16	16	16	16
6	celar09_pp_a	340	1130	7	16	16	16	16	16	16
7	graph05_pp_a	100	416	11	26	26	25	25	25	26
8	mildew	35	80	2	4	4	4	4	4	4
9	munin1	189	366	3	11	11	11	11	11	11
10	oesoca_hugina	67	208	9	11	11	11	11	11	11
11	oow_bas	27	54	2	4	4	4	4	4	4
12	oow_solo_a	40	87	4	6	6	6	6	6	6
13	oow-trad_a	33	72	4	6	6	6	6	6	6
14	pigs	441	806	3	10	10	10	10	10	10
15	ship-ship_a	50	114	4	8	8	8	9	8	9
16	vsd-hugin	38	62	2	4	4	4	4	4	4
17	water_a	32	123	9	10	10	9	10	9	10
18	wilson-hugin	21	27	2	3	3	3	3	3	3

Table 1 (a): The upper bounds produced by the heuristics described in this paper for some graphs from probabilistic networks and frequency assignment instances.

S	Graph name	|V|	|E|	LB	MF	EMF	MFEO1	MFEO2	RATIO1	RATIO2
1	anna	138	986	11	12	12	12	12	12	12
2	david	87	812	11	13	13	13	13	13	13
3	dsjc125_1	125	736	17	65	65	64	64	64	66
4	dsjc125_5	125	3891	55	111	111	110	110	110	109
5	dsjc250_1	250	3218	3	177	177	177	177	177	177
6	games120	120	1276	10	39	39	39	42	41	38
7	LE450_5A	450	5714	3	315	315	310	315	315	304
8	mulsol_i_4	175	3946	32	32	32	32	32	32	32
9	myciel4	23	71	10	11	11	10	10	10	10
10	myciel5	47	236	8	21	21	20	20	20	20
11	myciel6	95	755	20	35	35	35	35	35	35
12	myciel7	191	2360	31	66	66	66	66	66	66
13	queen5_5	25	320	12	18	18	18	18	18	19
14	school1	385	19095	80	225	225	225	225	225	209

Table 1 (b): The upper bounds produced by the heuristics described in this paper for some DIMACS vertex coloring instances.

The second aspect of interest of the implementation of the heuristics is the processing time. The EMF heuristic needs less time than that the other algorithms. Tables 2 (a) and 2

(b) give the details. Table 3 summarizes the behavior of the heuristics given in this paper on the graphs used in Table 1 (a) and Table 1(b). For each heuristic, it gives the number of times from each set of instances when the heuristic gives the best result, and when its result is smaller than the best result of all five heuristics.

S	Graph name	|V|	|E|	MF	EMF	MFEO1	MFEO2	RATIO1	RATIO 2
1	alarm1	37	65	0	0	0	0	0	0
2	barley	48	126	0	0	0	0	0	0
3	boblo	221	328	13	0	0	0	8	0
4	celar06_pp_a	100	350	1	0	0	0	1	0
5	celar07_pp_a	200	817	10	1	1	1	6	1
6	celar09_pp_a	340	1130	98	4	4	3	49	4
7	graph05_pp_a	100	416	1	0	1	1	0	1
8	mildew	35	80	0	0	0	0	0	0
9	munin1	189	366	9	0	1	1	5	1
10	oesoca_hugin_a	67	208	0	0	0	0	0	0
11	oow_bas	27	54	0	0	0	0	0	0
12	oow_solo_a	40	87	0	0	0	0	0	0
13	oow-trad_a	33	72	0	0	0	0	0	0
14	pigs	441	806	190	6	6	6	123	6
15	ship-ship_a	50	114	0	0	0	0	0	0
16	vsd-hugin	38	62	0	0	0	0	0	0
17	water_a	32	123	0	0	0	0	0	0
18	wilson-hugin	21	27	0	0	0	0	0	0

Table 2 (a): The used processing time in seconds for graphs from probabilistic networks and frequency assignment instances.

S	Graph name	|V|	|E|	MF	EMF	MFEO1	MFEO2	RATIO1	RATIO2
1	anna	138	986	2	0	0	0	0	0
2	david	87	812	0	0	0	0	0	0
3	dsjc125_1	125	736	7	6	6	6	6	6
4	dsjc125_5	125	3891	59	35	36	37	36	38
5	dsjc250_1	250	3218	478	451	451	451	453	441
6	games120	120	1276	3	2	2	1	2	2
7	LE450_5A	450	5714			13694	13437	15308	13301
8	mulsol_i_4	175	3946	29	23	27	26	26	28
9	myciel4	23	71	0	0	0	0	0	0
10	myciel5	47	236	0	0	1	0	0	0
11	myciel6	95	755	1	2	1	2	1	2
12	myciel7	191	2360	33	31	31	32	32	28
13	queen5_5	25	320	0	0	0	0	0	0
14	school1	385	19095	6280	5791	5742	5738	6390	3877

Table 2 (b): The used processing time in seconds for DIMACS vertex coloring instances.

S	Heuristic name	The upper bound produced by the heuristic is equal to the best one		The upper bound produced by the heuristic is worse than the best one	
		No. of graphs in set 1	No. of graphs in set 2	No. of graphs in set 1	No. of graphs in set 2
1	MF	16	7	2	7
2	EMF	16	7	2	7
3	MFEO1	18	10	0	4
4	MFEO2	17	10	1	4
5	RATIO1	18	11	0	3
6	RATIO2	16	13	2	1

Table 3: Comparison of heuristics: How often do the heuristics give or do not give the best result?

LB: Lower Bound.
MF: Minimum Fill-in.
ENF: Enhancement Minimum Fill-in.
MFEO1: Minimum Fill-in Excluding One neighbor vertex, version 1.
MFEO2: Minimum Fill-in Excluding One neighbor vertex, version 2.
RATIO1: The Ratio, version 1.
RATIO2: The Ratio, version 2.

4-2 Comparison heuristics introduced in this paper with known heuristics:

Tables 4-6 make a second type of comparison. Here, we compare the MFEO1 heuristic with a number of existing heuristics from the scientific literature. The data for the existing heuristics were taken from [6, 10]. Tables 4(a) and 4(b) give the upper bound values, Tables 5(a) and 5(b) the processing times, and Tables 6(a) and 6(b) an abstract comparison, in the same manner as Tables 1(a), 1(b), 2(a), 2(b), and 3. Each of these heuristics has been described in the overview [10]. The first five are based on building elimination order lists, and respectively use two variants of Lexicographic Breadth First Search (LEX-P and LEX-M), the Maximum Cardinality Search (MCS), the Minimum Fill-in heuristic (MF, also used in tables 1-3), and the Minimum Degree heuristic (MD: the vertex of minimum degree in the temporary graph is chosen). The last one is the Minimum Separating Vertex Sets heuristic (MSVS) from Koster [13], where a trivial tree decomposition is stepwise refined with help of minimum vertex separators.

S	Graph name	|V|	|E|	MFEO1	LEX-P	LEX-M	MF	MDFI	MCS	MSVS
1	alarm1	37	65	4	4	4	4	4	4	4
2	barley	48	126	7	7	7	7	7	7	7
3	boblo	221	328	3	4	4	3	3	3	3
4	celar06_pp_a	100	350	11	11	11	11	11	11	11
5	celar07_pp_a	200	817	16	19	18	16	18	18	18
6	celar09_pp_a	340	1130	16	19	18	16	18	19	18
7	graph05_pp_a	100	416	25	29	27	26	28	29	26
8	mildew	35	80	4	4	4	4	4	4	4
9	munin1	189	366	11	15	13	11	11	20	11
10	oesoca_hugin_a	67	208	11	12	11	11	11	11	11
11	oow_bas	27	54	4	4	4	4	4	5	4
12	oow_solo_a	40	87	6	6	6	6	6	6	6
13	oow-trad_a	33	72	6	6	6	6	6	6	6
14	pigs	441	806	10	19	18	10	10	15	15
15	ship-ship_a	50	114	8	9	9	8	8	9	9
16	vsd-hugin	38	62	4	4	4	4	4	5	4
17	water_a	32	123	9	10	10	10	11	10	10
18	wilson-hugin	21	27	3	3	3	3	3	3	3

Table 4 (a): The upper bounds for MFEO1 and heuristics from [10] for Probabilistic Networks and Frequency Assignment graphs.

S	Graphname	|V|	|E|	MF	EMF	MFEO1	MFEO2	RATIO1	RATIO2	D_LB
1	anna	138	986	12	12	12	12	12	12	12
2	david	87	812	13	13	13	13	13	13	13
3	dsjc125_1	125	736	65	65	64	64	64	66	67
4	dsjc125_5	125	3891	111	111	110	110	110	109	110
5	dsjc250_1	250	3218	177	177	177	177	177	177	176
6	games120	120	1276	39	39	39	39	41	38	41
7	LE450_5A	450	5714	315	315	310	315	315	304	323
8	mulsol_i_4	175	3946	32	32	32	32	32	32	32
9	myciel4	23	71	11	11	10	10	10	10	11
10	myciel5	47	236	21	21	20	20	20	20	20
11	myciel6	95	755	35	35	35	35	35	35	35
12	myciel7	191	2360	66	66	66	66	66	66	70
13	queen5_5	25	320	18	18	18	18	18	19	18
14	school1	385	19095	225	225	225	225	225	209	242

Table 4 (b): The upper bounds for D_LB from [6] and heuristics introduced in this paper for DIMACS vertex coloring instances.

S	Graph name	|V|	|E|	MFEO1	LEX-P	LEX-M	MF	MDFI	MCS	MSVS
1	alarm1	37	65	4	0	0	0	0	0	0
2	barley	48	126	7	0	0	0	0	0	0
3	boblo	221	328	3	1	10	0	0	1	0
4	celar06_pp_a	100	350	11	0	2	0	0	0	0
5	celar07_pp_a	200	817	16	2	16	0	0	2	3
6	celar09_pp_a	340	1130	16	0	0	0	0	0	0
7	graph05_pp_a	100	416	25	3	11	0	0	3	5
8	mildew	35	80	4	0	0	0	0	0	0
9	munin1	189	366	11	2	20	0	0	3	2
10	oesoca_hugin_a	67	208	11	0	0	0	0	0	0
11	oow_bas	27	54	4	0	0	0	0	0	0
12	oow_solo_a	40	87	6	0	0	0	0	0	0
13	oow-trad_a	33	72	6	0	0	0	0	0	0
14	pigs	441	806	10	14	161	0	0	8	10
15	ship-ship_a	50	114	8	0	0	0	0	0	0
16	vsd-hugin	38	62	4	0	0	0	0	0	0
17	water_a	32	123	9	0	0	0	0	0	0
18	wilson-hugin	21	27	3	0	0	0	0	0	0

Table 5 (a): The processing time for MFEO1 and heuristics from [10] for Probabilistic Networks and Frequency Assignment graphs.

S	Graphname	|V|	|E|	MF	EMF	MFEO1	MFEO2	RATIO1	RATIO2	D_LB
1	anna	138	986	3	0	0	1	1	1	1
2	david	87	812	0	0	1	0	0	0	0
3	dsjc125_1	125	736	7	6	6	6	6	6	3
4	dsjc125_5	125	3891			36	37	36	38	4
5	dsjc250_1	250	3218	478	451	451	451	453	441	33
6	games120	120	1276	3	2	2	1	2	2	2
7	LE450_5A	450	5714	10566	7340	13694	13437	15308	13301	274
8	mulsol_i_4	175	3946	29	23	27	26	26	28	14
9	myciel4	23	71	0	0	0	0	0	0	0
10	myciel5	47	236	0	0	1	0	0	0	0
11	myciel6	95	755	1	2	1	2	1	2	2
12	myciel7	191	2360	33	31	31	32	32	28	29
13	queen5_5	25	320	0	0	0	0	0	0	0
14	school1	385	19095	6280	5791	5742	5738	6390	3877	274

Table 5 (b): The processing time for D_LB from [6] and heuristics introduced in this paper for DIMACS vertex coloring instances.

LEX-P: Lexicographic Breadth First Search.
LEX-M: Lexicographic Minimum Triangulation.
MDFI: Minimum Degree Fill-in.
MCS: Minimum Cardinality Search.
MSVS: Minimum Separating Vertex Set.
D_LB: Heuristic from [6].

We conclude from the results of Tables 4, 5 and 6, that usually the best upper bounds were achieved by the MFEO1 heuristic. Table 6 (a) shows how often each of the heuristics MFEO1, LEX-P, LEX-M, MF, MCS, and D-LB gives the best result of all seven on the 18 graphs from probabilistic network and frequency assignment instances, and 14 graphs from DIMACS vertex-coloring instances. We can see that out 18 graphs of the first set of graphs in Table 4 (a), the upper bound of 2 graphs became better by using MFEO1 than that produced by the heuristics introduced in [10], the upper bounds of 14 graphs remain equal to the best upper bound of them, and no graph from this set of graphs its upper bound became worse by MFEO1 than that produced by those heuristics.

As well, by applying the MFEO1 on 14 graphs of the second set of graphs (DIMACS vertex-coloring instances), the upper bounds of 6 graphs became better than that produced by D_LB (introduced in [6]), upper bounds of 7 graphs remain equal to the best upper bound, and only for one graph, the upper bound obtained by D_LB method is better than that obtained by MFEO1. Table 6 (b) shows a comparison between MFEO1 and each of these seven heuristics. It gives number of graphs when MFEO1 gives a better, equal, or worse upper bounds than that produced by every heuristics introduced in [6, 10]. The processing time of the MFEO1 heuristic is relatively close to the processing time of other heuristics in spite of the fact that this algorithm uses $O(n^4)$ time in the worst case, while some of other heuristics are of $O(n^2)$ time complexity.

S	**Heuristic name**	**The upper bound produced by this heuristic is the only best one**		**The upper bound produced by this heuristic is equal to the best one**		**The upper bound produced by this heuristic is worse than the best one**		**Sum**	
		No. of graphs in set 1	**No. of graphs in set 2**	**No. of graphs in set 1**	**No. of graphs in set 2**	**No. of graphs in set 1**	**No. of graphs in set 2**	**No. of graphs in set 1**	**No. of graphs in set 1**
1	MFEO1	2	6	16	7	0	1	18	14
2	LEX-P	0		6		12		18	
3	LEX-M	0		7		11		18	
4	MF	0		3		15		18	
5	MD	0		11		7		18	
6	MCS	0		5		13		18	
7	D_LB		1		7	6			14

Table 6 (a): Comparison of heuristics: How often do the heuristics give or do not give the best result?

S	Heuristic name	The upper bound produced by MFEO1 is better	The upper bound produced by MFEO1 is equal	The upper bound produced by MFEO1 is worse	Sum
1	LEX-P	9	9	0	18
2	LEX-M	8	10	0	18
3	MF	16	2	0	18
4	MDFI	4	14	0	18
5	MCS	9	9	0	18
6	MSVS	6	12	0	18
7	D_LB	6	7	1	14

Table 6 (b): Comparison of heuristics: How often is MFEO1 better, equal or worse than every heuristics introduced in [6, 10]?

References:

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

[2] A. Berry, P. Heggernes, and G. Simonet. The minimum degree heuristic and the minimal triangulation process. *Proceedings of the 29th International Workshop on Graph-Theoretical Concepts in Computer Science, WG 2003*, pages 58-70, Elspeet, the Netherlands, 2003. Springer Verlag, Lecture Notes in Computer Science, vol. 2880.

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

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

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

[6] F. Clautiaux, J. Carlier, A. Moukrim, and S. Negre. New lower and upper bounds for graph treewidth. *Proceedings of the Second International Workshop on Experimental and Efficient Algorithms*, pages 70-80, Ascona, Switzerland, 2003. Springer Verlag, Lecture Notes in Computer Science, vol. 2647.

[7] M. C. Columbic. Algorithmic Graph Theory and Perfect Graphs. Academic Press, Inc., 1980.

[8] W. Cook and P. D. Seymour. Tour Merging via branch-decomposition. *Informs J. on Computing*, 15:233-248, 2003.

[9] A. Kornai and Z. Tuza. Narrowness, pathwidth, and their application in natural

language processing. *Discrete Application Mathematics*, 36:87-92, 1992.

[10] A. M. C. A. Koster, H. L. Bodlaender, and S. van Hoesel. Treewidth: Computational experiments. In Hajo Broersma, Ulrich Faigle, Johann Hurink, and Stefan Pickl (editors), *Electronic Notes in Discrete Mathematics*, volume 8. Elsevier Science Publishers, 2001.

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

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

[13] A. M. C. A. Koster. Frequency Assignment – Models and Algorithms. PhD thesis, Maastricht University, Maastricht, the Netherlands, 1999.

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

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

[16] K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. *Proceedings National conference on Artificial Intelligence (AAAI'97)*, pages 185-190. Morgan Kaufmann, 1997.

[17] J.A. Telle and R. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10:529-550, 1997.