Weighted Treewidth:
Algorithmic Techniques and Results

Emgad Bachoore

Hans L. Bodlaender

Department of Information and Computing Sciences,
Utrecht University

Technical Report UU-CS-2006-013

www.cs.uu.nl

ISSN: 0924-3275

Weighted Treewidth: Algorithmic Techniques and Results[⋆]

Emgad Bachoore and Hans L. Bodlaender

Department of Information and Computing Sciences,
Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
bachoore@cs.uu.nl hansb@cs.uu.nl

Abstract. From the analysis of algorithms for probabilistic networks, it is known that a tree decomposition of the minimum treewidth may not be optimal for these algorithms. Instead of treewidth, we consider therefore the weighted treewidth of a weighted graph. In this paper, we present a number of heuristics for determining upper and lower bounds on the weighted treewidth, and a branch and bound algorithm for finding the exact weighted treewidth for weighted graphs.

1 Introduction

In many graphical models and networks, each vertex is associated with a weight. The weights of the vertices in the graph or network play a significant role for finding an optimal solution for the problem. Triangulation of Bayesian Networks of probabilistic networks and logical partitioning approaches are examples of such problems.

Many decision support systems have probabilistic networks as underlying technology [10]. In these networks, we model dependencies and independencies between statistical variables using a directed acyclic graph. Each statistical variable is represented by a a vertex in the network. An important problem on these networks is probabilistic inference: we want to find the probability distribution for a variable, given a value assignment to some other variables. An efficient algorithm for inference is based on a tree decomposition of the moralized graph of the network, since this graph appears to have small treewidth for many probabilistic networks that model real-life situation. See for details [7–9].

In order to find a tree decomposition on which the algorithm from [8, 9] for inference takes little time, we search of a tree decomposition of small *weighted width*. The time this algorithm needs to process one bag of the tree decomposition is proportional to the product over the variables, represented by the vertices in the bag, of the number of different values that the variable can assume. If all values in the probabilistic network can assume the same number c of different values (e.g., all are binary and $c = 2$), then the time of the Lauritzen-Spiegelhalter algorithm is bounded by $O(c^k \cdot n)$. However, in practice, the statistical variables in a probabilistic networks may have different numbers of possible values, and thus the tree decomposition of minimum width may not be optimal for this algorithm. Thus, we look for a tree decomposition with minimum weighted width instead of one of minimum width.

The problem of finding the exact weighted treewidth of a weighted graph is an NP hard problem, even if all weights are equal [1]. Therefore, we introduce, in this paper, besides a weighted

[⋆] This research was supported by the project *Treewidth and Combinatorial Optimization* with a grant from the Netherlands Organization for Scientific Research NWO.

variant of the branch and bound algorithm for treewidth from [3], some heuristics for finding lower and upper bounds on weighted treewidth.

2 Preliminaries

We denote an undirected weighted graph by $G = (V, E, w)$ where V is the set of vertices of the graph, E is the set of edges of the graph and $w : V \rightarrow N^+$ is a weight function. We refer to the weight of a vertex v in a weighted graph G as $w_G(v)$, or in short $w(v)$.

We denote the set of neighbors of vertex v by $N(v)$, and the set of neighbors of v plus v itself by $N[v]$. 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 set of vertices W for which there is an $x \in W$ with $W - \{x\}$ a clique is called an ***almost clique***, x is called the ***excluding vertex***. A graph G is called ***triangulated*** (or: chordal) if every cycle of length four of more possesses a chord. A ***chord*** is an edge between two non consecutive vertices of the cycle. A graph $G = (V, E)$ is a subgraph of graph $H = (W, F)$ if $V \subseteq W$ and $E \subseteq F$. A graph $H = (V, F)$ 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 $f : V \rightarrow \{1, 2, \cdots, |V|\}$. A linear ordering of the vertices of a graph G, $\sigma = [v_1, \cdots, v_n]$ 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 $G[v_1, \cdots, v_n]$, i.e., the higher numbered neighbors of v_i form a clique. It has been shown in [8] that a graph G is triangulated, if and only if G has a $p.e.o.$

The weight of a set of vertices of a graph G, $S \subseteq V$ is $w(S) = \prod_{v \in S} w(v)$. The ***total weight of a graph*** G equals the weight of the set of its vertices, $w(V) = \prod_{v \in V} w(v)$. The ***neighborhood weight*** of a vertex v in a graph G, $nw_G(v) = \prod_{v \in N[v]} w(v)$, or in short $nw(v)$.

The definition of a tree decomposition of a weighted graph $G = (V, E, w)$ is exactly the same as for unweighted graphs. A ***tree decomposition*** of $G = (V, E, w)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a collection 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 each v, $\{i \in I \mid v \in X_i\}$ forms a connected subtree of T. The ***weighted width*** of a tree decomposition equals $\max_{i \in I} w(X_i)$ and the ***weighted treewidth*** of a graph G, $wtw(G)$, is the minimum weighted width over all tree decompositions of G.

A ***minor*** graph $G' = (W, F, w')$ of a weighted graph $G = (V, E, w)$ is a weighted graph obtained from G by a sequence of zero or more vertex removals, edge removals, and / or edge contractions, where an edge contraction for weighted graphs is the operation, that given an edge $\{x, y\} \in E$, removes x and y and their incident edges from G and adds a new vertex z, adjacent to the vertices that were adjacent to x or y, with the weight of z equal to $\min(w(x), w(y))$.

Lemma 1. *(See e.g., [4].)*

1. *For every triangulated graph $G = (V, E)$, there exists a tree decomposition $(X = \{X_i | i \in I\}, T = (I, F))$ 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 $W = X_i$.*

2. *Let $(X = \{X_i | i \in I\}, T = (I, F))$ be a tree decomposition of G of width at most k. The graph $H = (V, E \cup E')$, with $E' = \{\{v, w\} | \exists i \in I : v, w \in X_i\}$, obtained by making every set X_i a clique, is triangulated, and has maximum clique size at most $k + 1$.*
3. *Let $(X = \{X_i | i \in I\},\ T = (I, F))$ 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 [6, 7]). The weighted treewidth of a graph G is the minimum $k \geq 0$ such that G is a subgraph of a triangulated graph with all cliques of weight at most k.*

Lemma 3. *(See [7]). Let G and G' be two weighted graphs. If G' be a minor of G, then the weighted treewidth of G', $wtw(G')$, is at most the weighted treewidth of G, $wtw(G)$.*

Definition 1. *The* fill-in *of a vertex x in a weighted graph G,* fill-in$_G(x)$*, or in short,* fill-in(x)*, is the number of edges that must be added between the neighbors of x to make it simplicial, i.e., the neighborhood of that vertex turn into a clique.*

$$\text{fill-in(x)} = |\{\{v, w\} | v, w \in neighbors(x), \{v, w\} \notin E\}|$$

Definition 2. *The* fill-in *excluding one neighbor of a vertex x in a weighted graph G,* fill-in-excl-one$_G(x)$*, or in short,* fill-in-excl-one(x)*, is the* minimum *number of edges that must be added between the neighbors of x to make it almost simplicial, i.e., by adding these edges to the graph, the neighborhood of that vertex will turn into an almost clique.*

$$\text{fill-in-excl-one(x)} = \min_{z \in neighbors(x)} |\{\{v, w\} | v, w \in neighbors(x) - \{z\}, \{v, w\} \notin E\}|$$

3 Algorithms for finding weighted treewidth

In this section, we first introduce two heuristics for obtaining a lower bound on the weighted treewidth. Then we introduce a number of heuristics for obtaining an upper bound on weighted treewidth. Finally, we present the weighted variant of the branch and bound algorithm for treewidth from [3].

3.1 Lower Bound Heuristics for Weighted Treewidth

In the following two lemmas, we give two simple lower bounds on the weighted treewidth of a graph. Later, we generalize two known heuristics for a lower bound on the treewidth, namely, Maximum Minimum Degree and the weighted variant of the Ramachandramurthi graph parameter.

Lemma 4. *Let G be a weighted graph, $wtw(G) \geq \max_{v \in V}(w(v))$.*

Proof. Let v be a vertex of maximum weight in a weighted graph G and let $(\{X_i | i \in I\}, T = (I, F))$ be a tree decomposition of G of minimum weighted width. By the definition of a tree decomposition of a weighted graph, there is at least one node $i \in I$, such that $v \in X_i$. Therefore, the weighted treewidth of G is at least the weight of v. $\square$

Lemma 5. *(Eijkhof et al. [7]). Let G be a weighted graph. $wtw(G) \geq \min_{v \in V}(nw(v))$.*

The Maximum Minimum Neighborhood Weight Heuristic, variant Lower Bound (MMNW_lb) This heuristic builds upon Lemma 5. Given a weighted graph $G = (V, E, w)$. The algorithm works as follows: In the first step, the lower bound on the weighted treewidth is initialized with zero, $lb = 0$. Then we repeat the following operations until the graph becomes empty: Let $v \in V$ be the vertex of the minimum neighborhood weight in G; set the lower bound on the weighted treewidth to the maximum of its current value and the minimum neighborhood weight in the graph, $lb \leftarrow max(lb, nw(v))$, and remove v and its incident edges from G. In Figure 1, we give the pseudo-code of this algorithm.

```
MMNW_lb Algorithm($G$)
    Input: A weighted graph $G = (V, E, w)$;
    Output: A lower bound on weighted treewidth of a graph $G$, $lb$;
1   let $G' = (V', E') = G = (V, E)$, $lb = 0$;
2   while ($G'$ is not empty)
3       let $v$ be a vertex with $nw_{G'}(v)$ the minimum amongst all vertices in $G'$;
4       set $lb \leftarrow \max(lb, nw_{G'}(v)$;
5       set $V' \leftarrow V' - \{v\}$; $E' = E' - \{v, w\}, w$ is a neighbor of $v$ in $G'$;
6   return($lb$);
```

Fig. 1. Pseudo-code of the Maximum Minimum Neighborhood Weight (MMNW) Algorithm .

Lemma 6. *The weighted treewidth of a weighted graph G is at least the lower bound obtained from the MMNW_lb heuristic, applied to G.*

The Weighted $\gamma_w(G)$ Parameter Heuristic Ramachandramurthi [11] introduced the graph parameter $\gamma(G)$. Let G be an unweighted graph, $\gamma(G) = \min(n - 1,$ $\min_{v,w \in V, \{v,w\} \notin E}(\max(degree(v), degree(w)))$, i.e., $\gamma(G) = n - 1$, if G is a clique.

Lemma 7. *(See Ramachandramurthi [11]). For every graph G, $tw(G) \geq \gamma(G)$.*

The weighted variant of γ, γ_w is defined as follows.

Definition 3. *For each weighted graph G,*

$$\gamma_w(G) = \min(\prod_{v \in V}(w(v)), \min_{v,w \in V, \{v,w\} \notin E} \max(nw(v), nw(w)))$$

Note that $\gamma_w(G) = \prod_{v \in V}(w(v))$, if G is a clique.

Lemma 8. *For every weighted graph G, $wtw(G) \geq \gamma_w(G)$.*

Proof. If G is a complete graph (clique), then $\gamma_G = \prod_{v \in V}(w(v)) = wtw(G)$. Otherwise, let $x = wtw(G)$ and let v, w be a pair of nonadjacent vertices of $weight \leq x$ in G. Then $x \geq \max(nw(v), nw(w)) \geq \gamma_w(G)$. □

The Weighted $\gamma_w(G)$ Heuristic consists of two main steps. In the first step, we also initialize the lower bound on the weighted treewidth with zero, i.e., $lb = 0$. In the second step, if the graph is a clique, then we set lb to the maximum of its current value and the weight of the graph. Otherwise, we compute the minimum over all edges of the maximum weight of an endpoint of the edge. We set lb to the maximum of its current value and this minimum, and repeat on the graph G', obtained by removing the vertex that yielded this minimum value and its incident edges. As we compute at each step $\gamma_w(G')$ for a subgraph of G, we obtain a lower bound on the weighted treewidth of G. The pseudo-code of this algorithm is given in Figure 2.

```
$\gamma_w(G)$ **Algorithm**$(G)$
    **Input:** A weighted graph $G = (V, E, w)$;
    **Output:** A lower bound on weighted treewidth of graph $G$, $lb$;
1   **let** $G' = (V', E') \leftarrow G = (V, E), lb \leftarrow 0$;
2   **while** ($G'$ is not empty)
3       **if** ($G'$ is a clique)
4           **return**$(\max(w(G') = \prod_{v \in V}(w(v)), lb))$;
5       **else**
6           $CurGam \leftarrow +\infty$;
7           **foreach** $v, w \in V'$, $\{v, w\} \notin E'$
8               **if** $(nw(v) \geq nw(w))$ **then** $e \leftarrow v$; **else** $e \leftarrow w$;
9               **if** $(nw(e) \leq maxInt)$ **then** $CurGam \leftarrow nw(e)$; $x \leftarrow e$;
10          **set** $V' \leftarrow V' - \{x\}$; $E' = E' - \{x, w\}, w$ is a neighbor of $x$ in $G'$;
11          **if** $((CurGam > lb)$ **then** $lb \leftarrow CurGam$;
```

Fig. 2. Pseudo-code of the $\gamma_w(G)$ Algorithm.

3.2 Upper Bound Heuristics for Weighted Treewidth

The following lemma gives a very primitive upper bound on the weighted treewidth of the graph.

Lemma 9. *The weighted treewidth of a weighted graph G is at most $\prod_{v \in V} w(v)$.*

In the following, we present a number of heuristics for the upper bound on the weighted treewidth. All these heuristics depend basically on building a triangulation for a given graph. The following lemma is a weighted variant of a well known result for treewidth, and can be proved in the same way as the unweighted case, see e.g., [4].

Lemma 10. *Let G be a triangulated graph. The weighted treewidth of G equals the maximum weight over all maximal cliques $Q = (W, F)$ in G of $\prod_{w \in W} w(v)$.*

For building a triangulation $H_\sigma = (V, F, w)$ for a weighted graph $G = (V, E, w)$, one can use a linear ordering σ of the vertices of G such that σ is a perfect elimination ordering (p.e.o.) of H_σ, in the following way. For $i = \sigma[1], \cdots, \sigma[|V|]$, 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. Thus, σ is a p.e.o. of the resulting graph H_σ. As H_σ is triangulated, its weighted treewidth equals the maximum weight of the set of vertices $S \subseteq V$, whose vertices form a maximal clique in H_σ. See Lemma 10. Hence, the weighted treewidth of H_σ, $wtw_{H_\sigma} = \prod_{v \in S} w(v)$. The following result can also be derived in the same way as the unweighted case.

Lemma 11. *There is at least one linear ordering σ for a weighted graph $G = (V, E, w)$ where we obtain the exact weighted treewidth of G.*

This suggests the general scheme in Figure 3 for the upper bound heuristics on the weighted treewidth.

```
**set** $G' \leftarrow G$; $i \leftarrow 1$; $\sigma \leftarrow ()$; $ub \leftarrow 0$;
**while** $G'$ is not the empty graph
    **select** according to some condition a vertex $v$ from $G'$;
    **set** $ub \leftarrow \max(ub, nw(v_{G'}))$;
    **eliminate** $v$; /* remove $v$ and turn its neighbors into a clique */
    **add** $v$ to position $i$ in the ordering $\sigma$;
    **set** $i \leftarrow i + 1$;
{Now $ub$ is an upper bound on the weighted treewidth of $G$.}
```

Fig. 3. A general scheme for the upper bound heuristics on the treewidth

We call a graph G' encountered during the algorithm a temporary graph. Thus, the main differences between the following heuristics are in their conditions for selecting a vertex v from G', at each step where we have to eliminate a vertex from the graph. Therefore, we will limit our discussion over these heuristics by giving the selection conditions of each heuristic. The following two lemmas are the common factors between the selection conditions of these heuristics.

Lemma 12. *(See [7]). Let v be a simplicial vertex in a weighted graph $G = (V, E, w)$. Then the weighted treewidth of G is at least the neighborhood weight of v, $nw(v)$.*

Definition 4. *We call to a vertex v in a weighted graph G a **strongly almost simplicial** vertex in G if and only if it is almost simplicial with $N(v) - \{x\}$ a clique, the neighborhood weight of v is at most the weighted treewidth of G, $nw(v) \leq wtw(G)$, and the weight of x is at most the weight of v, $w(x) \leq w(v)$.*

Lemma 13. *(See [7]). Let v be a weighted strongly almost simplicial vertex in a weighted graph $G = (V, E, w)$. Then the weighted treewidth of G is at least the neighborhood weight of v, $nw(v)$.*

In [5], the notion of ***safe reduction rule*** was introduced. A safe reduction rule rewrites a graph G to a smaller one, G', 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 $\max(low, wtw(G)) = \max(low', wtw(G'))$. The *Simplicial and Strongly Almost Simplicial Rules* are examples of safe rules that we have used in the selection conditions of our heuristics.

The Maximum Minimum Neighborhood Weight Heuristic, Upper Bound variant (MMNW_ub) The main steps of this heuristic are the same as in MMNW_lb with one step more. Namely, before we remove a vertex from the graph, we add an edge between every two non adjacent neighbors of that vertex. In other words, we make a clique from the neighborhood of that vertex. Hence, the selection conditions we used in this heuristic are based upon selecting at each step when eliminate a vertex from a temporary graph, the vertex with the minimum neighborhood weight in this graph.

Lemma 14. *The weighted treewidth of a graph $G = (V, E, w)$ is at most the output of the MMNW heuristic, applied to G.*

The Minimum Fill-in Heuristic, Weighted Variant (MF_W) In this variant of the Minimum Fill-in Heuristic, we compute an upper bound on the weighted treewidth in the same manner as in the MMNW_ub heuristic, with one exception: in the selection conditions of this heuristic, we select the vertex with minimum fill-in in the temporary graph instead of the vertex with the minimum neighborhood weight as in MMNW_ub.

Lemma 15. *The weighted treewidth of a graph $G = (V, E, w)$ is at most the output of the MF_W heuristic, applied to G.*

The Minimum Fill-in Excluding One Heuristic, variant Weighted (WMFEO) We have developed three versions of this heuristic. The differences between these are in the sequences of the conditions we use for selecting the vertex we have to eliminate from the temporary graph. In all three versions of this heuristic, and at any step when we have to eliminate a vertex from the temporary graph, we check if this graph contains any simplicial or strongly almost simplicial vertices. We eliminate these vertices from the graph, if they exist, and set the upper bound on the weighted treewidth to the maximum of its current value and the maximum neighborhood weight of these vertices. Otherwise, depending on the version, we select a version as follows. In the first version, WMFEO1, we perform this check: Let p be a vertex with minimum fillin in a temporary graph $G' = (W, F)$ of a given graph G. We select a vertex $q \in W$ such that, $q \neq p$, *fill-in-excl-one(q)* $\leq$ *fill-in(p)*, $nw(q) \leq low$, and $w(x) \leq w(q)$, where x is the excluded neighbor of q and low is a lower bound on the weighted treewidth of G. If more than one vertex q satisfies to these conditions, then we select the vertex of the minimum *fill-in* among them, but if still there is more than one vertex with these specifications, then we select the first vertex of the minimum *fill-in-excl-one* among these.

In version 2 of the algorithm, WMFEO2, the ties are broken using the fill-in and fill-in-excl-one in the reverse order. If more than one vertex q satisfies to the two conditions, namely, *fill-in-excl-one(q)* $<$ *fill-in(p)*, $nw(q) < low$ and $w(x) \leq w(q)$, then the vertex with minimum fill-in amongst these is eliminated first. But, if there still is more than one vertex that satisfies the last condition, then the vertex with minimum *fill-in-excl-one* amongst these should be processed first.

In version 3 of the algorithm, WMFEO3, the ties are broken using the neighborhood weight besides the fill-in and fill-in-excl-one. If more than one vertex q satisfies to the two conditions, *fill-in-excl-one(q)* $<$ *fill-in(p)*, $nw(q) < low$ and $w(x) \leq w(q)$, then the vertex with minimum neighborhood weight amongst these is eliminated first.

Lemma 16. *Let ub be the upper bound on the weighted treewidth obtained from applying WMFEO1, WMFEO2, or WMFEO3 to a graph $G = (V, E, w)$. The weighted treewidth of G is at most ub.*

The Ratio Heuristic, Weighted Variant (WRATIO) We have adapted the two versions of Ratio heuristic introduced in [2], to be used for computing an upper bound on the weighted treewidth of a weighted graph. The rules we use for selecting a vertex that we should eliminate are as follows: Again, as long as there are safe vertices in the temporary graph, namely, simplicial and/or strongly almost simplicial vertices, we eliminate these first, and set the upper bound to the maximum of its current value and the maximum neighborhood weight of these vertices. After that, each version of the heuristic proceeds in a different way. In version 1, we proceed as follows: Let p be a vertex with the minimum *fill-in* in the temporary graph G' of a graph G. A vertex $w \neq p$ in the temporary graph is selected if the *fill-in-excl-one* of w is less than or equal to the *fill-in* of p, its neighborhood weight is at most a lower bound on the weighted treewidth of G, $nw(v) \leq wtw(G)$, the $w(v) \leq w(x)$, where x is the excluded neighbor of w, and it satisfies the following condition. Let $r_1(w) =$ *fill-in-excl-one(w) / fill-in(p)*, and $r_2(w) = nw(w)/nw(p)$. We now require that $r_1(w) < r_2(w)$ to be a candidate for selection at this point. If we have more than one such candidate, we select from these a vertex with the minimum difference between r_1 and r_2, $(r_1 - r_2)$.

In version 2 of this heuristic, we proceed as follows: For all $w \in W(G')$, we select the vertex of the minimum ratio $r(w) =$ *fill-in(w)* / $nw(w)$, $(nw(w) > 1)$ amongst all vertices of G'.

Lemma 17. *Let ub be the upper bound on the weighted treewidth obtained from applying WRATIO (version 1 or version 2) to a graph $G = (V, E, w)$. The weighted treewidth of G is at most ub.*

We end this subsection with some general observations.

Lemma 18. *Let G be a complete weighted graph, $wtw(G) = \prod_{v \in V}(w(v))$.*

Proof. In Lemma 1, (3), we see that if $(X = \{X_i | i \in I\},\ T = (I, F))$ is a tree decomposition of G of minimum weighted width and $W \subseteq V$ form a clique in G. Then there exists an $i \in I$ with $W \subseteq X_i$. Thus, the weighted treewidth of G is at least $\prod_{v \in V}(w(v))$. In Lemma 9, we see that the weighted treewidth of G is at most $\prod_{v \in V}(w(v))$. Hence, the weighted treewidth of G equals $\prod_{v \in V}(w(v))$. □

Lemma 19. *Let $G = (V, E, w)$ be a triangulated weighted graph. The weighted treewidth of G equals the upper bound obtained from the MMNW_ub heuristic.*

Proof. We use induction to $|V|$. The result clearly holds when $|V| = 1$. Suppose that v is the first vertex selected. Thus, $nw(v) = \min_{w \in V} nw(w)$. Let G' be the temporary graph, obtained from eliminating v from G, i.e., G' is obtained by turning the neighborhood of v into a clique and then removing v and its incident edges.

G' is again a triangulated graph. Consider a cycle C of length at least four in G'. If each edge in C is an edge in G, then clearly C has a chord, as G is triangulated. Suppose x, y are successive vertices on C with $\{x, y\}$ an edge in G' but not in G. Then x and y are neighbors of v. Consider the cycle C' in G, obtained by inserting v between x and y in C. This cycle must have a chord in G: this chord is either the edge $\{x, y\}$ (impossible by assumption), also a chord in G', or of the form $\{v, z\}$ for some z on C. As C has length at least four, either x or y is not successive to z on C, say x. Then the edge $\{x, z\}$ is added to G' when eliminating v, so again C has a chord. Thus, we have shown that each cycle in G' of length at least four has a chord, thus G' is triangulated.

We can observe that the MMNW_ub heuristic outputs the maximum of the neighborhood weight of v and the result of running the MMNW_ub heuristic on G'. Let w be a simplicial vertex in G. We have $nw(v) \leq nw(w)$, and the weighted treewidth of G is at least $nw(w)$, by the simpliciality of w. As G' is triangulated, the MMNW_ub heuristic gives the weighted treewidth of G', i.e., the maximum weight over all maximal cliques Q in G'.

Thus, it remains to show that for each (maximal) clique Q in G', the weight is at most the weighted treewidth of G. If Q is also a clique in G, then this clearly holds. Suppose now that Q is not a clique in G. Then $Q' = Q \cap N(v) \neq \emptyset$. We consider two cases.

– $Q' = Q$. I.e., all vertices in Q belong to $N(v)$. Then the weight of Q is at most $nw(v)$, hence at most the weighted treewidth of G.
– $Q' \neq Q$. Then there is a vertex $z \in Q$, $z \notin N(v)$. For each pair of vertices $x, y \in Q'$, $x \neq y$, look at the cycle v, x, z, y. This is a cycle in G of length four in G, and as $z \notin N(v)$, $\{x, y\} \in E$. But now we have that v is simplicial, and so no edges are added when eliminating v, and thus Q is also a clique in G.

□

Lemma 20. *Let $G = (V, E)$ be a triangulated graph. The weighted treewidth of G equals the upper bound obtained from the following upper bound heuristics:*

1. *The MF_W heuristic.*
2. *The WMFEO heuristic.*
3. *The WRATIO heuristic.*

Proof. This follows as these algorithms select at each step a simplicial vertex. □

3.3 Improving Upper and Lower Bound Heuristics

In order to obtain better lower and upper bounds on the weighted treewidth from the above heuristics, it is wisely to incorporate some safe rules in the *selection conditions* of these heuristics. One can do that as follows. At each step when we have to select and eliminate a vertex for the graph using some specific selection conditions, we choose the vertices that do not cause a worse upper or lower bound than that we will obtain when we select other vertices. An example for such safe rules is the rule of selecting simplicial and strongly almost simplicial vertices whenever they exist. The question that could be arise now is the following. Is it worthwhile to spent some time to test, at each state, whether or not there are simplicial or strongly almost simplicial vertices in the graph? Theoretically, it has been proved that selecting of these vertices is safe (see Lemma's 12 and 13). Practically, the results of our experiments reported in Section 4 and the results reported in [7] support this.

3.4 A branch and bound algorithm for weighted treewidth

After we have introduced a number of heuristics for determining lower and upper bounds on weighted treewidth in the previous sections, we introduce in this section a weighted variant of branch and bound algorithm, *BB-tw* that we have introduced in [3], for computing the exact weighted treewidth of weighted graphs. The goals for developing this algorithm were: First, to determine the exact weighted treewidth of some graphs, in particular for graphs with at most 50 vertices. Second, to be able to more precisely establish the quality of the given upper and lower bound, as an exact algorithm allows us to compare the outcome of these heuristics with the exact values. Third, we can use branch and bound algorithm for improving the upper and lower bounds on the weighted treewidth, obtained from the above heuristics, as we have described in [3], if determining the exact weighted treewidth is not possible within a reasonable time.

In the weighted variant of branch and bound algorithm, W-BB-tw, we have the same space of all feasible solutions as that we have described for unweighted variant. Briefly, this space consists of all possible elimination orderings of the vertices of the given graph. The input to the algorithm are a weighted graph $G = (V, E, w)$, the best known upper and lower bounds, obtained from the heuristics for the weighted treewidth of G described in this paper, and a perfect elimination ordering that gave the best upper bound that is known. The algorithm works as follows: At the beginning, we check whether the best upper bound, ub, equals the best lower bound, lb, obtained from the upper and lower bound heuristics. If so, then the algorithm return this value as the exact weighted treewidth of G. Otherwise, we test every (apart from pruning) possible elimination orderings, in the space of all feasible solutions, whether the elimination of the vertices of the graph due to this order produces an exact weighted treewidth or a better upper bound than reported so far. Moreover, we prune any solution in that space, which delivers an upper bound that is greater than or equal to the reported one so far. The steps for eliminating the vertices of the graph and producing a triangulation of G for each elimination ordering are as it is described in Figure 3.

The pruning rules that we have incorporated in this variant of the algorithm are similar to those we used for unweighted variant in [3]. We have adapted all the pruning rules we have described for unweighted variant to be used for the weighted variant. The following two Lemmas show the differences in the methods of computing the treewidth of a graph and the weighted treewidth of a weighted graph.

Lemma 21. *Let $H_1, \cdots, H_r$, $r \geq 1$, be all possible triangulated graphs of a graph G. Let $Q_1, \ldots, Q_r$ be the cliques of maximum size in $H_1, \cdots, H_r$. Then, the treewidth of G equals the minimum size of $Q_1, \cdots, Q_r$.*

Proof. Let H_i be a triangulated graph of G, $Q_i = (W, F)$ be a clique of the maximum size in H_i, $D_i = \{(X_i|i \in I, T = (I, F)\}$ be a tree decomposition of H_i. Then the treewidth of H equals $|W| - 1$ (Lemma 1). The treewidth of G equals the minimum width of all possible tree decompositions of G minus 1 (def. of the treewidth). Therefore, the treewidth of G equals the minimum size of the maximum cliques of all possible triangulated graphs of G. □

Lemma 22. *Let $H_1, \cdots, H_r$, $r \geq 1$, be all possible triangulated graphs of a weighted graph G. Let $Q_1, \ldots, Q_r$ be the cliques of maximum weight in $H_1, \cdots, H_r$. Then, the weighted treewidth of G equals the minimum weight of $Q_1, \cdots, Q_r$.*

Proof. The proof of this lemma is similar to the proof of Lemma 21. Also here, if we suppose that H_i be a triangulated graph of G, $Q_i = (W, F)$ be a clique of the maximum weight in H_i, $D_i = \{(X_i|i \in I, T = (I, F)\}$ be a tree decomposition of H_i. Then the weighted treewidth of H equals $w(W) = \prod_{v \in W} w(v)$. The weighted treewidth of G equals the minimum weighted width over all possible tree decompositions of G (def. of the weighted treewidth). Therefore, the weighted treewidth of G equals the minimum of the maximum weights of the cliques over all possible triangulated graphs of G. □

The differences in the manners in which the pruning rules can be used in both variants of the problem follow from the differences in these two characterizations of treewidth and weighted treewidth. Consider the pruning rules given in Section 3.2 of [3]. Some of these pruning rules have to be modified when we consider the weighted variant, while the others remain as in the unweighted variant. Below, we discuss the rules that are modified for the weighted case. The rules that are not changed can be found in [3].

Pruning Rule 2: The weight of the temporary graph Let G' be the temporary graph obtained from eliminating a set of vertices X from a given weighted graph G. Let max be the maximum neighborhood weight of all vertices $x \in X$, at the step when they were eliminated from G and added to X. If the total weight of the vertices in G' is less than or equal to the value of max, then we replace the upper bound value reported so far with the value of max, and prune the subtree rooted at the last vertex, $x \in X$, that has been eliminated from G and added to the set X, from the space of all feasible solutions.

The rule becomes as follows in the weighted variant of the BB-tw algorithm.

```
Pruning Rule 2:
let $G = (V, E, w)$ be a weighted graph
    $(ub, lb)$ be the best upper and lower bound on the $wtw(G)$,
    reported so far,
    $G' = (W, F, w)$ be the weighted graph after eliminating a set of
    vertices $X = \{x | x \in V, x \notin W\}$ and their incident edges from $G$,
    $maximum = \max_{x \in X}(nw(x))$,
    $y$ be the last vertex that has been eliminated from $G$,
    $w(G') = \prod_{w \in W} w(w)$;
if $(w(G') \leq maximum)$
    $ub = maximum$;
    omit the subtree rooted at the parent of vertex $y$ from the
    space of all feasible solutions;
```

Lemma 23. *Let $G' = (W, F, w')$ be the weighted graph obtained from eliminating a vertex y from a weighted graph $G = (V, E, w)$ such that, for each $w \in W$, $w'(w) = w(w)$. Let $r = nw(y)$ at the step when y is eliminated from G. If $\prod_{w \in W} w(w) < r$, then the treewidth of G is at most r.*

Proof. Eliminating the vertices of the graph G' in any order and reporting the maximum of these neighborhood seen during the process will not cause the treewidth of the graph to become larger than r since $r \geq \prod_{w \in W} w(w)$. □

Pruning Rule 3: The weight of the eliminated vertex We check in this rule whether the neighborhood weight of the vertex that we have to eliminate, $nw(v)$, is greater than or equal to the best upper bound on the weighted treewidth reported so far. If such a case holds, then the current elimination ordering will not generate a better upper bound on the weighted treewidth than the one we have reported right now. Therefore, we prune this elimination ordering from the space of all feasible solutions and continue the search operation for the exact weighted treewidth or a better upper bound in the next elimination ordering.

```
Pruning Rule 3
let $v$ be the current vertex we have to eliminate from a graph $G$,
    $ub$ be the best upper bound on the weighted treewidth of $G$ we have
    reported so far;
if $nw(v) \geq ub$
    omit the current elimination ordering from the space of all feasible
    solutions;
```

Lemma 24. *Let $H = (W, F, w)$ be a weighted triangulation of a weighted graph $G = (V, E, w)$ and ub be an upper bound on the weighted treewidth of G. If $\exists w \in W$, $nw(w) > ub$ and w is simplicial, then $wtw(G) < wtw(H)$.*

Proof. Since w is simplicial in H, the weighted treewidth of H is at least the neighborhood weight of w, $nw(w)$ (Lemma 12). Now, we have that $nw(w)$ is greater than an upper bound on the weighted treewidth of G. Thus, the weighted treewidth of G is less than the weighted treewidth of H. □

Pruning Rule 5: Simplicial and strongly almost simplicial vertices In the Pruning Rule 5 of the branch and bound algorithm BB-tw for unweighted graphs introduced in [3], if the graph contains any simplicial or a strongly almost simplicial with a degree at most the best upper bound reported so far, then we eliminate this vertex or these vertices from the graph and prune all the elimination orderings from the space of all feasible solutions which, the values of their elements equal the values of their correspond elements of the current elimination ordering up to this position.

In the weighted variant of the algorithm, this rule is a rather straightforward generalization of the unweighted variant in the case of simplicial vertices. But, in the case of almost simplicial vertices, a vertex should fulfil to the following conditions to be strongly almost simplicial: The vertices in its neighborhood form an almost clique, its neighborhood weight is at most the best upper bound reported right now, and its weight is at most the weight of the excluding vertex from its neighbor.

```
Pruning Rule 5
let $v$ be the current vertex we have to eliminate from a graph $G$;
if $v$ is simplicial or $v$ is strongly almost simplicial
    omit all the subtrees rooted at the parent of $v$ from the space of all feasible
    solutions excluding the subtree rooted at $v$;
```

4 Computational Experiments

In this section, we report on computational experiments for the seven heuristics for computing an upper bound on weighted treewidth, introduced in Section 3.2, the two heuristics for computing a lower bound on the weighted treewidth, introduced in Section 3.1, and the branch and bound algorithm for computing the exact weighted treewidth for a graph, BB-tw, introduced in Section 3.4.

All algorithms were implemented using C++ on a Windows 2000 PC with Pentium 4, 2.8 GHz processor. The tables shown in this section include besides the basic information for the graph, also columns for the treewidth of the graph (tw), the upper bound (ub), the lower bound (lb) on the weighted treewidth and the running time of the algorithm (t). We use the character ”*” in the column t to indicate that the algorithm did not terminate normally, namely, the algorithm did run out of time. We defined three hours as the maximum time limit for running the BB-tw algorithm on the input graph, i.e., if the algorithm did not find the exact treewidth within three hours, then it was ended and returned an upper bound value for the treewidth.

Each table we give in this section includes 15 instances of sizes between 21 and 441 vertices, and between 27 and 806 edges. The Alarm, Oesoca, Vsd and Wilson are probabilistic networks taken from medical applications; several versions exist of the Myciel networks. The Barley and

Mildew networks are used for agricultural purposes, the Water network models a water purification process and Oow-trad, Oow-bas, Oow-solo, and Ship-ship networks are developed for maritime use. The other graphs are obtained from the well-known DIMACS benchmarks for vertex coloring.[1]

In Table 1, we show the results of the implementation of the BB-tw algorithm, weighted variant, on this set of graphs. We observe that BB-tw algorithm, weighted variant, was able to determine the exact weighted treewidth for all instances of sizes less than 50 vertices except one, namely, Ship-ship, within a running time between 0.0009 and 7640.44 seconds.

Graphname	$\|V\|$	$\|E\|$	best lb	best ub	tw	t
Alarm	37	65	32	32	32	0.00
Barley	48	126	151200	1.22472e7	6.3504e6	3115
Mildew	35	80	280000	1.7568e6	805200	0.16
Oesoca	39	67	240	240	240	0.00
Oesoca42	42	72	240	240	240	0.00
Oesoca+	67	208	5760	92160	69120	22.472
Oow-bas	27	54	18270	822150	510300	3.240
Oow-solo	40	87	18270	3.402e6	2.916e6	7640.44
Oow-trad	33	72	18270	3.1789e7	2.48472e6	4734.76
Ship-ship	50	114	56700	1.2096e8	6.2208e7	*
VSD	38	62	240	360	360	0.00
Water	32	123	16384	1.76947e6	589824	0.381
Wilson	21	27	108	108	108	0.00

Table 1. Results of the WBB-tw algorithm

[1] See http://www.cs.uu.nl/people/hansb/treewidthlib.

Graphname	Size		ub Heuristic						
			MMNW_ub	MF_W	WMFEO1	WMFEO2	WMFEO3	WRATIO1	WRATIO2
	$\|V\|$	$\|E\|$							
alarm	37	65	32	32	32	32	32	32	32
barley	48	126	1.30637e7	1.30637e7	**1.224727e7**	**1.22472e7**	1.30637e7	1.30637e7	3.08629e11
mildew	35	80	2.81088e6	**1.7568e6**	**1.7568e6**	**1.7568e6**	**1.7568e6**	**1.7568e6**	3.08629e11
munin1	189	366	7.84e7	7.84e7	7.84e7	7.84e7	7.84e7	7.84e7	2.86654e19
oesoca	39	67	240	240	240	240	240	240	240
oesoca42	39	67	240	240	240	240	240	240	240
oesoca+	67	208	138240	138240	**92160**	138240	**92160**	138240	138240
oow_bas	27	54	1.5309e7	**822150**	**822150**	**822150**	**822150**	**822150**	6.561e7
oow_solo	40	87	3.0618e7	1.701e7	**3.402e6**	1.31544e7	**3.402e6**	4.3848e6	3.13905e11
oow_trad	33	72	1.97316e7	1.27159e8	7.45416e7	4.9329e7	7.45416e7	**3.17898e7**	6.43743e10
pigs	441	806	177147	177147	177147	177147	177147	177147	2.28768e13
ship-ship	50	114	2.4192e8	8.4672e8	**1.2096e8**	**1.2096e8**	8.4672e8	8.4672e8	7.52716e17
VSD	38	62	360	360	360	360	360	360	360
water	32	123	5.30842e6	1.76947e6	1.76947e6	1.76947e6	1.76947e6	1.76947e6	1.35895e9
wilson-hugin	21	27	108	108	108	108	108	108	108

Table 2. The upper bounds obtained from the upper bound heuristics introduced in Section 3.2

Graphname	Size		ub Heuristic						
			MMNW_ub	MF_W	WMFEO1	WMFEO2	WMFEO3	WRATIO1	WRATIO2
	$\|V\|$	$\|E\|$							
alarm	37	65	0.00	0.00	0.00	0.00	0.00	0.00	0.00
barley	48	126	0.90	0.83	0.97	0.88	0.88	0.80	0.26
mildew	35	80	0.02	0.02	0.04	0.07	0.06	0.06	0.05
munin1	189	366	0.04	0.07	0.09	0.12	0.08	0.15	0.09
oesoca	39	67	0.00	0.00	0.00	0.00	0.00	0.00	0.00
oesoca42	39	67	0.00	0.00	0.00	0.00	0.00	0.00	0.00
oesoca+	67	208	0.01	0.13	0.11	0.15	0.16	0.19	0.11
oow_bas	27	54	0.00	0.00	0.00	0.00	0.00	0.03	0.00
oow_solo	40	87	0.06	0.94	0.86	0.57	0.0.75	1.21	0.74
oow_trad	33	72	0.00	0.05	0.06	0.12	0.07	0.17	0.01
pigs	441	806	1.34	2.04	2.29	2.76	2.66	2.80	1.77
ship-ship	50	114	0.08	0.16	0.11	0.34	0.58	0.81	0.17
VSD	38	62	0.00	0.00	0.00	0.00	0.00	0.00	0.00
water	32	123	0.00	0.00	0.03	0.00	0.01	0.09	0.00
wilson-hugin	21	27	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 3. The running times of the upper bound heuristics, introduced in Section 3.2, on a set of graphs

In Table 2, we show the results of implementing the upper bound heuristics, introduced in Section 3.2 on the same set of graphs.

We notice clearly from the results in these tables the following: First, the upper bounds of 5 instances, obtained from these heuristics, equal the exact weighted treewidths of these instances, where the best lower bounds of all these five instances equal their best upper bounds. Second, the upper bounds obtained from the three versions of WMFEO are better than or equal to those obtained from MMNWF as well as from MF_W for all instances of Table 2. Third, the upper bounds of two instances, namely, Oesoca+ and Oow-solo, obtained from WMFEO, version 1 and 3 are better than the upper bounds obtained from all other upper bound heuristics for the same graphs. Fourth, the upper bound of only one graph, namely, Oow-trad, obtained from Ratio1 is better than that obtained from all other upper bound heuristics for the same graph. Five, the upper bound obtained from Ratio2 heuristic are the worst amongst all other upper bound heuristic, for all graphs of Table 2. The running time of implementing the upper bound heuristics on this set of graphs are given in Table 3.

Finally, in Table 4, we give the results of implementing the two lower bound heuristics, introduced in Section 3.1, for the weighted treewidth of a weighted graph. The lower bounds of 5 instances, obtained from one of these heuristics or both of them, equal to their exact weighted treewidth. We notice that the gaps between the lower bounds obtained from these heuristics and the exact weighted treewidths for many of these graphs are very large, specially when the lower bound does not equal to the lower bound on the weighted treewidth of a graph. For many instances, MMNW_lb heuristic performs better than $\gamma_w(G)$ heuristic. This is because in MMNW_lb heuristic, we select a vertex that satisfies to the safe preprocessing rules before we select any other vertex, while we do not do that in $\gamma_w(G)$ heuristic.

Graphname	Size		ub	lb Heuristic			
				MMNW_lb		$\gamma_w(G)$	
	$\lvert V\rvert$	$\lvert E\rvert$		lb	t	lb	t
alarm	37	65	32	32	0	32	1
barley	48	126	1.224727e7	**151200**	0	100800	1
mildew	35	80	1.7568e6	280000	0	280000	0
munin1	189	366	7.84e7	**3600**	0	1280	1.07
oesoca	39	67	240	240	0	240	0
oesoca42	42	72	240	240	0	240	0
oesoca+	67	208	69120	**8640**	0	1920	0
oow_bas	27	54	822150	**28594**	0	18270	0
oow_solo	40	87	3.402e6	**24300**	0	18270	0
oow_trad	33	72	3.17898e7	**28350**	0	18270	0
pigs	441	806	177147	**729**	1	81	43
ship-ship	50	114	1.2096e8	**107520**	0	56700	0
VSD	38	62	360	360	0	240	0
water	32	123	1.76947e6	16384	0	16384	0
wilson-hugin	21	27	108	108	0	54	0

Table 4. Results of the lower bound heuristics introduced in Section 3.1

Acknowledgements

We would like to thank Arie Koster for valuable suggestions, and helpful comments.

References

1. 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.
2. E. H. Bachoore and H. L. Bodlaender. New upper bound heuristics for treewidth. In S. E. Nikoletseas, editor, *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms WEA 2005*, pages 217–227. Springer Verlag, Lecture Notes in Computer Science, vol. 3503, 2005.
3. E. H. Bachoore and H. L. Bodlaender. A branch and bound algorithm for exact, upper, and lower bounds on treewidth. Submitted, 2006.
4. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc.*, 209:1–45, 1998.
5. 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.
6. F. v. d. Eijkhof and H. L. Bodlaender. Safe reduction rules for weighted treewidth. In L. Kučera, editor, *Proceedings 28th Int. Workshop on Graph Theoretic Concepts in Computer Science, WG'02*, pages 176–185. Springer Verlag, Lecture Notes in Computer Science, vol. 2573, 2002.
7. F. v. d. Eijkhof, H. L. Bodlaender, and A. M. C. A. Koster. Safe reduction rules for weighted treewidth. To appear in Algorithmica.
8. F. V. Jensen. *Bayesian Networks and Decision Graphs*. Statistics for Engineering and Information Science, Springer-Verlag, New York, 2001.
9. 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.
10. J. Pearl. *Probablistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Palo Alto, 1988.
11. S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM J. Disc. Math.*, 10:146–157, 1997.