Alcom-FT Technical Report Series
ALCOMFT-TR-03-123

Safe reduction rules for weighted treewidth

Frank van den Eijkhof

Hans L. Bodlaender

Arie M.C.A. Koster

institute of information and computing sciences, utrecht university

technical report UU-CS-2002-051

www.cs.uu.nl

Safe reduction rules for weighted treewidth*

Frank van den Eijkhof[†] Hans L. Bodlaender[‡] Arie M.C.A. Koster[§]

Abstract

Several sets of reductions rules are known for preprocessing a graph when computing its treewidth. In this paper, we give reduction rules for a weighted variant of treewidth, motivated by the analysis of algorithms for probabilistic networks. We present two general reduction rules that are safe for weighted treewidth. They generalise many of the existing reduction rules for treewidth. Experimental results show that these reduction rules can significantly reduce the problem size for several instances of real-life probabilistic networks.

1 Introduction

For many graph problems, it is useful and important to find a tree decomposition of minimal width (called treewidth) [4, 11, 13]. Often these problems can be solved in linear or polynomial time when a tree decomposition of bounded width is known.

The problem of finding a tree decomposition with minimum width is NP-hard [1], and approximating treewidth is also NP-hard [10]. In [6] it was shown that preprocessing techniques can help reducing the size of instances of these problems. A set of reduction rules is given that can be used to preprocess a graph for the computation of its treewidth. Each of these rules reduces the number of vertices of the graph. Rules for which a tree decomposition for the reduced graph with minimum treewidth can easily be extended to a tree decomposition for the original graph that also has minimum treewidth are called safe.

*The work of the first author was done while he was working at the Institute of Information and Computing Sciences, Utrecht University, with partial support by the Netherlands Computer Science Research Foundation with financial support from the Netherlands Organisation for Scientific Research. The second author was partially supported by EC contract IST-1999-14186: Project ALCOM-FT (Algorithms and Complexity – Future Technologies). This work was partially carried out in the project *Treewidth and Combinatorial Optimization* with financial support from the Netherlands Organisation for Scientific Research.

†Department of Methodology and Statistics, Faculty of Social Sciences, Utrecht University, P.O. Box 80.0140, 3508 TC Utrecht, the Netherlands. Email: f.vandeneijkhof@fss.uu.nl.

‡Institute of Information and Computing Sciences, Utrecht University. P.O. Box 80.089, 3508 TB Utrecht, the Netherlands.

§Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, D-14195 Berlin-Dahlem, Germany.

To allow more safe rules, a variable *low* is maintained that invariantly is a lower bound on the treewidth of the graph. A reduction rule is now called *safe*, if and only if the maximum of *low* and the treewidth of the graph is not changed. More results on reduction algorithms for graphs of bounded treewidth can be found in [2, 8].

In this paper, we generalise the results of [6] to a weighted variant of treewidth. The problem is motivated from the area of probabilistic networks, but it is also useful for combinatorial optimisation problems. Several modern decision support systems have *probabilistic networks* as underlying technology [15]. These networks model dependencies and independencies between statistical variables using a directed acyclic graph. For each statistical variable, represented by a vertex in the graph, a probabilistic function is defined. The most important problem to solve on these networks is *probabilistic inference*, which computes the probability distribution of a variable given a value-assignment to other variables. The most efficient algorithm currently used for probabilistic inference is based upon the use of a tree decomposition of the so-called moralised graph of the network, since this graph appears to have small treewidth for many probabilistic networks that model real-life situations. We refer to e.g., [9, 13] for details.

A tree decomposition consists of a number of 'bags', organised in a tree. (For the precise definition, see Section 2.) The treewidth is one smaller than the maximum size of a bag. The time the algorithm from [9, 13] takes to process one bag is proportional to the product over the variables represented by vertices in the bag of the number of different values they can attain. So, e.g., if we have a probabilistic network with all variables having the same number c of values (for instance, $c = 2$ for binary variables), then the time to execute the algorithm based on a tree decomposition with n bags and of width k becomes bounded by $O(c^k \cdot n)$. In such a case, one may want to find a tree decomposition of minimum treewidth to be used by the probabilistic inference algorithm. However, the statistical variables in a probabilistic network may have different numbers of values, and thus a tree decomposition with minimum treewidth may not be optimal for this algorithm. Instead of treewidth, we therefore consider the weighted treewidth problem, where we try to find tree decompositions such that the maximum over all bags of the product of the weights of the vertices in the bag is as small as possible. We notate the weight of vertex v by $w(v)$; this is the finite number of values the variable associated with v can attain. The weighted treewidth expresses the maximum product of weights of the bags in a tree decomposition (note that when all vertices have weight c, the treewidth of a graph is one less than the logarithm (with base c) of the weighted treewidth).

When preprocessing a graph, regardless we consider weighted or unweighted treewidth, we repeatedly apply safe reduction rules from some set, until no more rules from the set can be applied. After construction of a tree decomposition for the remaining graph, undoing the reductions in reverse order gives us a tree decomposition for the original graph. In case the remaining graph is empty, the optimal weighted treewidth is computed this way. In case the remaining graph is not empty, whether the optimal weighted treewidth is obtained depends on the technique used to produce a tree decomposition for the reduced graph. An exact, but possibly computationally expensive algorithm (like integer linear programming) guarantees a tree decomposition with optimal weighted width for the original graph. By using heuristic algorithm(s) only an approximation of the weighted treewidth is obtained. This approximation becomes exact, although no exact algorithm is used, when the weighted width computed for the reduced graph is at most the value *low*.

In Section 3, we present two rules for preprocessing weighted treewidth. The first is the Simplicial rule, which is a rather straightforward generalisation of the non-weighted case. It removes a simplicial vertex, which is a vertex for which all neighbours form a clique, and updates a variable *low* representing the lower bound for the weighted treewidth of the original graph. The second rule comprises many possible reduction rules. It is called the *Contraction Reduction rule*. For this rule we try to generate a clique that separates some single vertices from the rest of the graph using *contraction* of edges. Besides the safeness for this rule, we show that the detection whether the contraction reduction rule can be applied is in general NP-complete. In Section 4, we consider several special cases of the Contraction Reduction rule that can be identified in polynomial time. Unweighted variants of these instances, specially the Almost Simplicial, Buddy and (Extended) Cube rules have been introduced and proven safe for unweighted treewidth previously in [4, 6]. We thus have generalised these rules to the weighted case. In addition, we obtain new rules which are safe for weighted treewidth. The Buddies rule generalises the Buddy rule and is an example of such a new rule.

As for the unweighted case, the contraction reduction rule bases on a variable *low* that invariantly is a lower bound on the weighted treewidth. The higher *low* is, the more effective the rule can be. To increase *low* in practice, we discuss in Section 5 lower bounds for weighted treewidth. Several experiments conducted by applying the reduction rules on 23 real-life probabilistic networks and lower bounds are presented in Section 6. Our experiments reveal that for several networks a decomposition with minimal weighted width can be found, while most remaining networks are reduced significantly. Some concluding remarks close the paper.

2 Definitions and preliminaries

In this paper, we assume that graphs are undirected, simple, and have a weight function $w : V \to \mathbf{N}^+$. We use the notation $G = (V, E, w)$.

We assume familiarity with standard graph notions, like independent set, clique, etc.

Let $G = (V, E, w)$ be a graph. With $\bar{G}$ we define the complement of G. The set of neighbours of a vertex v is denoted $N(v)$. Note that we assume that $v \notin N(v)$. The *degree* of a vertex is denoted as $deg(v) = |N(v)|$. A vertex $v \in V$ is *simplicial* when $N(v)$ induces a clique. A *subgraph* $H(G)$ of G is a graph $H = (V', E', w[V'])$ with $V' \subseteq V$, $E' \subseteq (V' \times V') \cap E$, and $w[V'] : V' \to \mathbf{N}^+$ a function that assigns for every vertex $v \in V'$ the value $w(v)$ to $w[V'](v)$. For a set of vertices $W \subseteq V$, the *subgraph induced by* W is the subgraph $G[W] = (W, (W \times W) \cap E, w[W])$. For the sets of vertices $W, X \subset V$, W and X disjoint, the set X *separates* W when every path between a vertex in W and a vertex in $V \backslash (W \cup X)$ uses a vertex in X.

Let $G = (V, E, w)$ be a graph. The weight of a set of vertices $S \subseteq V$ is $w(S) =$

$\prod_{v \in S} w(v)$. The *neighbourhood weight* or $nw_G(v)$ of a vertex $v \in V$ is $nw_G(v) = w(N(v) \cup \{v\})$. When G is clear from the contest, we write $nw(v)$.

A *tree decomposition* of a graph $G = (V, E)$, or a weighted graph $G = (V, E, w)$, is a pair $(\{X_i | i \in I\}, T = (I, F))$ with T a tree and for every $i \in I$ a *bag* $X_i \subseteq V$, such that for each vertex $v \in V$ there exists a bag with $v \in X_i$, for each edge $\{v, u\} \in E$ there exists a bag with $v, u \in X_i$, and for each vertex $v \in V$ the induced graph $T[S_v]$, with $S_v = \{i \in I | v \in X_i\}$, is a tree.

The *width* of a tree decomposition $(\{X_i | i \in I\}, T = (I, F))$ of a (weighted) graph $G = (V, E, w)$ equals $\max_{i \in I} |X_i| - 1$; the *treewidth* of a graph G, denoted $\tau(G)$ is the minimum width over all tree decompositions of G. Similar the *weighted width* equals $\max_{i \in I} w(X_i)$ (without minus 1!) and the *weighted treewidth* of a graph G, denoted $\tau_w(G)$, is the minimum weighted width over all tree decompositions of G.

Let $G = (V, E, w)$ be a graph. A *contraction* of an edge $\{v, u\} \in E$ with $w(v) \leq w(u)$ makes v adjacent to $(N(v) \cup N(u)) \backslash \{v\}$, and removes vertex u and edge $\{v, u\}$. Rephrased more intuitively, an edge is contracted to the incident vertex that has the smallest weight.

A *minor* of a graph G is a graph G' that is obtained from G by a sequence of zero or more vertex removals, edge removals, and/or edge contractions.

Lemma 1 *Let G' be a minor of G. Then $\tau_w(G') \leq \tau_w(G)$.*

Proof. (This proof is similar to that for the unweighted case, see e.g. [5, Lemma 16].) Let $(\{X_i | i \in I\}, T = (I, F))$ be a tree decomposition of $G = (V, E)$ of optimal width. If G' is obtained from G by removing an edge $e \in E$, then $(\{X_i | i \in I\}, T)$ is also a tree decomposition of G. If G' is obtained from G by removing a vertex $v \in V$, then $(\{X_i - \{v\} | i \in I\}, T)$ is a tree decomposition of G. If G' is obtained from G by contracting the edge $\{x, y\} \in E$ to vertex x with $w(x) \leq w(y)$, then $(\{X'_i | i \in I\}, T = (I, F))$ with for all $i \in I$, $X'_i = X_i$ when $y \notin X_i$ and $X'_i = X_i - \{y\} \cup \{x\}$ when $y \in X_i$, is a tree decomposition of G'. In each case, the weighted treewidth of G' is at most the treewidth of G. The result now follows by induction. □

3 General reduction rules

In this section we define two general reduction rules. Both rules concentrate upon deletion of vertices that can be separated from the graph by cliques. The first rule deletes simplicial vertices, and the second rule is based upon sets of edges that can be contracted such that they form cliques in the graph. It's unweighted version generalises some reduction rules known for (unweighted) treewidth. In the next section we discuss these special cases in the context of weighted treewidth.

When reversing a reduction, for each vertex deleted in the reduction a new bag is added containing this vertex and its neighbours. Because the neighbours form a clique in the reduced graph, the bag can be easily added to the tree decomposition. We must be able to guarantee that this bag will not increase the weighted treewidth to a value above the weighted treewidth of the original graph. Therefore we maintain a variable *low* that represents the largest lower bound we know for the weighted treewidth of the original graph.

A reduction rule is called *safe* when application of the rule changes a graph G and its associated variable low_G, into G' and its associated variable $low_{G'}$, such that $\max(low_G, \tau_w(G)) = \max(low_{G'}, \tau_w(G'))$. Safeness of a reduction rule implies that when we start with *low* having any value that is at most the weighted treewidth of the original graph, the rule does not increase the weighted treewidth of the graph above that of the original graph. In Section 5 we discuss a heuristic for setting an initial *low*.

3.1 The Simplicial Rule

The Simplicial rule is a straightforward generalisation of the same rule of the unweighted case, and easily follows from well-understood properties of chordal graphs and tree decompositions.

> **The Simplicial rule**
> Let v be a simplicial vertex in graph $G = (V, E, w)$.
> Set *low* to $\max(low, nw(w))$.
> Remove v and its incident edges from G.

Theorem 2 *The Simplicial rule is safe.*

Proof. As G contains a clique of weight $nw(v)$, we know that $\tau_w(G) \geq nw(v)$. Furthermore, because $G - v$ is a minor of G, we know that $\tau_w(G) \geq \tau_w(G - v)$. Therefore, $\tau_w(G) \geq \max(nw(v), \tau_w(G - v))$.

Now let T' be a tree decomposition for $G - v$ with weighted width $k \leq \max(nw(v), \tau_w(G - v))$. We create a tree decomposition T by adding a bag X_i to T' as follows. Bag X_i consists of $N(v) \cup \{v\}$, and we connect X_i to a bag X_j in T' with $i \neq j$ and $N(v) \subseteq X_j$. Since $N(v)$ is a clique in G', X_j exists ([7, Lemma 3.1]). The weighted width of T now equals $\max(nw(v), k)$, and thus $\tau_w(G) \leq \max(nw(v), \tau_w(G - v))$, hence $\tau_w(G) = \max(nw(v), \tau_w(G - v))$. □

In Figure 1 an application of the Simplicial rule is illustrated. Solid lines represent edges, and the dotted lines connect to the remainder of the graph. The numbers represent the weights on the vertices. For this example, *low* will become at least 120.

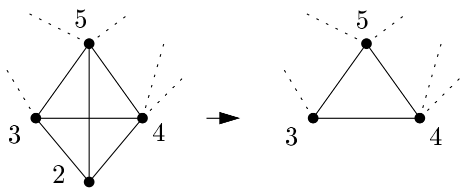

Figure 1: An instance of the *Simplicial rule*.

3.2 The Contraction Reduction Rule

Next, we introduce the general Contraction Reduction rule and show that it is safe. We also show that it is NP-complete to determine for a given graph G whether it is possible to carry out the Contraction Reduction rule in G. Fortunately, there are several special instances of the rule for which there are efficient algorithms that determine if and where the rule can be carried out in a given graph; several of these cases will be discussed in Section 4. We first define the notion of *contraction reduction*.

Definition A *contraction reduction* in a weighted graph $G = (V, E, w)$ is a 3-tuple (X, Y, S) with $X,\ Y \subset V$ disjoint non-empty sets of vertices, and $S \subseteq E$ a set of edges, with the following properties:

1. X is an independent set.
2. Each edge in S has one endpoint in X and one endpoint in Y.
3. A vertex in X is incident to exactly one edge in S.
4. For each $x \in X$, $N(x) \subseteq Y$.
5. For each $\{x, y\} \in S$, $x \in X$, $y \in Y$: $w(x) \geq w(y)$.
6. Contraction of all edges in S will turn Y into a clique.

See Figure 2 for an example of a contraction reduction; the edges in S are drawn by fat lines. An equivalent way of stating the last condition of a contraction reduction is that for every pair of vertices $y, y' \in Y$, either $y = y'$, $\{y, y'\} \in E$, or there is an $x \in X$ with $\{x, y\} \in S$ and $\{x, y'\} \in E$ or $\{x, y\} \in E$ and $\{x, y'\} \in S$. There are some special cases of contraction reductions that are 'uninteresting', e.g., when there is a simplicial vertex $x \in X$ for contraction reduction (X, Y, S). In order not to make the definition more complex, we do not explicitly forbid this case.

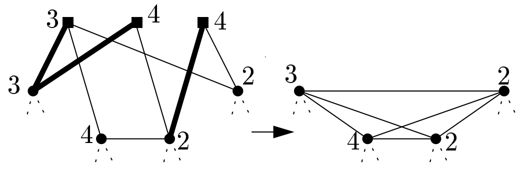

Figure 2: A contraction reduction

The Contraction Reduction rule
Let (X, Y, S) be a contraction reduction in $G = (V, E, w)$.
If $\quad low \geq \max_{x \in X} nw(x)$,
then Remove all vertices of X from G.
Turn Y into a clique.

Next, we show the safeness of the Contraction Reduction rule.

Theorem 3 *The Contraction Reduction rule is safe.*

Proof. Let G, low be given. Suppose (X, Y, S) is a contraction reduction in G, and $low \geq \max_{x \in X} nw(x)$. Let G' be the graph, resulting from the application of the rule. Note that G' is obtained from G by contracting all edges in S, so G' is a minor of G, and hence $\tau_w(G') \leq \tau_w(G)$. Combined with the precondition $low \geq \max_{x \in X} nw(x)$, we have that $\max(low, \tau_w(G)) \geq \max(\tau_w(G'), \max_{x \in X} nw(x))$.

Now let T' be a tree decomposition of G'. We obtain T from T' by adding a bag X_x to T' for each vertex $x \in X$ as follows. Bag X_x consists of $N(x) \cup \{x\}$, and we connect X_x to a bag X_j in T' with $x \neq j$ and $N(x) \subseteq X_j$. Since $N(x) \subseteq Y$ is a clique in G', X_j exists (see [7, Lemma 3.1]).

The weighted width of T is at most the maximum of $\tau_w(T')$ and $\max_{x \in X} nw(x)$, and thus $\tau_w(G) \leq \max(\tau_w(G'), \max_{x \in X} nw(x))$. Hence, $\tau_w(G) = \max(\tau_w(G'), \max_{x \in X} nw(x))$. So $\max(\tau_w(G), low) = \max(\tau_w(G'), low)$. □

Note that the contraction reduction in Figure 2 is safe when $low \geq 72$. As a byproduct of the proof of Theorem 3, we have a method to 'undo' the rules: if we have a tree decomposition of the reduced graph G' of treewidth not more than that of the original input graph, then we can construct a tree decomposition of the graph just before the reduction step by applying the construction given in the proof. The resulting tree decomposition will have weighted width at most the maximum of low and $\tau_w(G)$.

Next we show that it is NP-complete to determine for a given graph G (and value low) whether it is possible to apply the Contraction Reduction rule for G.

Theorem 4 *It is NP-complete to decide if there exists a contraction reduction in a given graph $G = (V, E, w)$, even if we require that w is a constant function.*

Proof. Clearly, the problem belongs to NP. To proof that it is NP-hard, we reduce from Vertex Cover. In the Vertex Cover problem, we are given a graph $G = (V, E)$ and an integer $k \leq |V|$, and ask if there is a set of vertices $W \subseteq V$ with $|W| \leq k$, such that every edge $e \in E$ has at least one endpoint in W.

Let a graph $G = (V, E)$, and an integer k be given. Without loss of generality, assume $|V| \geq 5$ and $k \geq 3$. Let $G' = (V', E')$ be obtained from G in the following way. Suppose $V = \{v_1, v_2, \ldots, v_n\}$. We take the complement of G, add k new vertices $\{u_1, \ldots, u_k\}$ that are made adjacent to each vertex in V, and add n new vertices $z_1, \ldots, z_n$, making z_i adjacent to z_{i-1}, z_{i+1}, and v_i, with z_1 adjacent to z_n. See figure 3.

We write $U = \{u_1, \ldots, u_k\}$ and $Z = \{z_1, z_2, \ldots, z_n\}$. Now, $V' = V \cup U \cup Z$, and $E' = \{\{y, z\} \mid y, z \in V,\ y \neq z,\ \{y, z\} \notin E\} \cup \{\{u_i, y\} \mid u_i \in U,\ y \in V\} \cup \{\{v_i, z_i\} \mid 1 \leq i \leq n\} \cup \{\{z_i, z_{i+1}\} \mid 1 \leq i < n\} \cup \{\{z_n, z_1\}\}$. Assume $w(y) = 2$ for all $y \in V'$.

We now claim that G has a vertex cover of size at most k, if and only if G' has a contraction reduction. As G' can be constructed in polynomial time from G, the theorem follows from this claim.

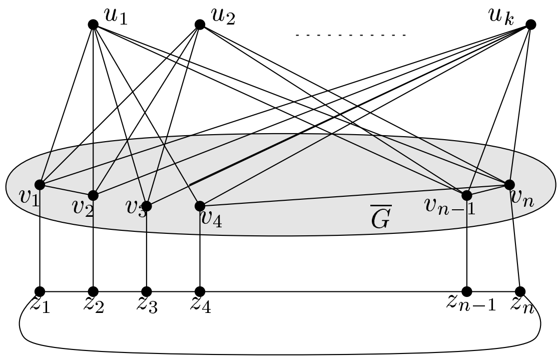

Figure 3: Graph G' in the proof of Theorem 4

Suppose $W = \{w_1, \ldots, w_{k'}\}$ is a vertex cover of size at most k in G, $k' \leq k$. If $k' < k$, set $w_{k'+1}, \ldots, w_k$ to arbitrary vertices in V. Set $S = \{\{u_i, w_i\} \mid 1 \leq i \leq k\}$. Now, $(\{u_1, \ldots, u_k\}, V, S)$ is a contraction reduction in G'. Most conditions of contraction reduction hold by construction. We show that the contraction of edges in S turns V into a clique. Consider $y, z \in V$, $y \neq z$, either $\{y, z\} \in E'$, or otherwise, $\{y, z\} \in E$ and then $y \in W$ or $z \in W$ as W is a vertex cover. Without loss of generality, suppose $y = w_i$ for some i, $1 \leq i \leq k$. u_i is contracted to y, and as u_i is adjacent to z, the edge $\{y, z\}$ is created by the contraction. So, V forms a clique after contraction of all edges in S.

Suppose that G' has a contraction reduction (X, Y, S).

First, assume that $X \cap U \neq \emptyset$. Suppose $u_i \in X \cap U$. As $N(u_i) = V$, $V \subseteq Y$. Let W be the set of vertices in S that are endpoint of an edge in S with the other endpoint in U. It follows that $|W| \leq k$. We claim that W is a vertex cover of G. Consider a pair $y, z \in V$ with $\{y, z\} \in E$. Now $\{y, z\} \notin E'$, so the edge $\{y, z\}$ must be obtained from contracting the edges in S. As a vertex in Z is adjacent to exactly one vertex in V, $\{y, z\}$ cannot be obtained from a contraction with a vertex in Z, and hence either $y \in W$ or $z \in W$. So we have a vertex cover of G of size at most k.

Now, assume that $X \cap U = \emptyset$. This case will lead to a contradiction. Suppose $X \cap V \neq \emptyset$. As u_1 and u_2 are adjacent to each vertex in V, $\{u_1, u_2\} \subseteq Y$, and hence there must be an edge of the form $\{v_i, u_j\} \in S$, $j \in \{1, 2\}$, $1 \leq i \leq n$. Then $\{u_3, z_i\} \subseteq N(v_i) \subseteq Y$. To create the edge $\{u_3, z_i\}$ by contracting S, there must be a common neighbour of u_3 and of z_i that is contracted to u_3 or z_i, but their only common neighbour is already contracted to u_j, contradiction. So we conclude that $X \cap V = \emptyset$, and hence $X \subseteq Z$.

Suppose $z_i \in X$. To ease notation, suppose $1 < i < n$. We have that $\{z_{i-1}, z_{i+1}, v_i\} \subseteq Y$. As z_{i-1} and z_{i+1} have only z_i as common neighbour, z_i must be contracted to one of its neighbours in Z. As this argument applies to every vertex in $X \cap Z$, all endpoints of edges in S belong to Z. This yields a contradiction: either the edge $\{v_i, z_{i-1}\}$ or the edge $\{v_i, z_{i+1}\}$ is not obtained by contraction of the edges in S. This finishes the proof that G has a vertex cover of size at most k, and hence our NP-completeness proof. □

Consider the problem to decide, given a graph $G = (V, E, w)$ and a value for *low*, if G can be reduced to a smaller graph with the Contraction Reduction rule. The proof above shows that this problem is NP-complete, even in the case that $w(v) = 2$ for all vertices $v \in V$ and $low = 2^{n+1}$.

3.3 Confluence of the rules

Since several simplicial and/or contraction reduction rules can be applicable at the same time for a graph, naturally the question rises whether the order the rules are applied is of influence on the graph obtained in the end. We conjecture that this is not the case. For multiple applications of the simplicial rule this is easy to see. For the contraction reduction rule and the combination of the rules it seems to be harder to derive the result.

Conjecture 5 (Confluence of rules) *The order in which the simplicial rule and the contraction reduction rule are applied to a graph does not influence the remaining graph.*

4 Specific reduction rules

As discussed in the previous section, it is NP-complete to determine for a given graph whether the Contraction Reduction rule can be used to reduce its size. Fortunately, there are several special cases that have efficient algorithms to find out whether and where they can be applied. We first discuss these special cases which generalise known results for the unweighted case, providing sufficient conditions on the weights of vertices and the value of *low* to make these rules safe.

In [3] Arnborg and Proskurowski identified a complete set of reduction rules for (unweighted) graphs with treewidth at most three. In other words: at least one of these rules can be applied to any non-empty graph of treewidth at most three, and applying the rule will not turn a graph with treewidth at most three in one of treewidth more than three, and vice versa. Thus, the rules can be used to obtain an algorithm to recognise graphs of treewidth at most three; see also [14]. This set of rules consists of the Islet, Twig, Series, Triangle, Buddy, and (Extended) Cube rule. The Islet and Twig rule are instances of the Simplicial rule, expressing the cases that the simplicial vertex has degree 0 or 1. In [6], the Almost Simplicial rule was shown to be safe for (unweighted) treewidth. This rule generalises the Series and Triangle rule. We will show below that the Almost Simplicial rule is a special case of the Contraction Reduction rule. Also, we will show that the Buddy and (Extended) Cube rules are also special cases of the Contraction Reduction rule. In each case, this gives a proof of safeness of generalisations of these rules to the weighted case, and establishes conditions upon the weights when the rules can be carried out. Sanders [16] extended the set of rules from [3] to a much larger set of rules that is safe and complete for non-weighted graphs with treewidth at most four. However, only a subset of his reduction rules are instances of our Contraction Reduction rule; many of his rules are not reversible easily, see also [12].

We define the Almost Simplicial rule as follows.

The Almost Simplicial rule

Let u and v be vertices in graph $G = (V, E, w)$.
Suppose that v is a neighbour of u.
Suppose that $N(v)\backslash\{u\}$ forms a clique in G.
If $\quad low \geq nw(v)$ and $w(v) \geq w(u)$,
Then Turn $N(v)$ into a clique
Remove v and its incident edges from G.

Corollary 6 *The Almost Simplicial rule is safe.*

Proof. Safeness of the Almost Simplicial rule follows from the fact that it is a special case of the Contraction Reduction rule. Let G, u, v be as in the rule given above, with $w(v) \geq w(u)$. Then $(\{v\}, N(v), \{\{v, u\}\})$ is a contraction reduction in G. The condition that $low \geq \max_{x \in X} nw(x)$ from the Contraction Reduction rule is here equivalent to $low \geq nw(v)$. Thus, by Theorem 3, the Almost Simplicial rule is safe. □

Figure 4 shows an instance of the Almost Simplicial rule; it can be carried out when $low \geq 288$. The dashed lines indicate potential edges.

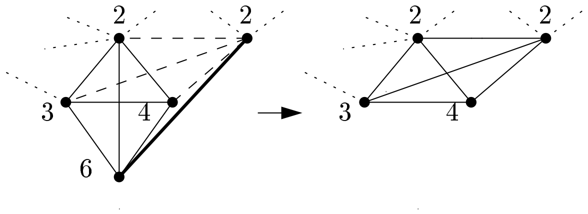

Figure 4: An instance of the *Almost Simplicial rule*.

The Series and Triangle rules are special cases of the Almost Simplicial rule. The Series rule is the case that v has degree two. So, given a vertex v of degree two, the Series rule removes v and adds an edge between its neighbours when the weight of v is at least the weight of one of its neighbours, and low is at least the product of the weights of v and the weights of its neighbours. The Triangle rule is the case that v has degree three in the Almost Simplicial rule.

We next define the Buddies rule. In the Buddies rule, we have r vertices, each with the same $r + 1$ neighbours; when certain conditions on the weights and low hold, we can remove these r vertices and turn the set of $r + 1$ neighbours into a clique.

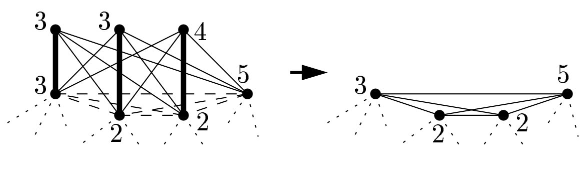

Figure 5: An instance of the *Buddies rule*.

The Buddies rule
Let r be a positive integer.
Let $X = \{x_1, \ldots, x_r\}$ be a set of r vertices in graph $G = (V, E, w)$.
Let $Y = \{y_1, \ldots, y_{r+1}\}$ be a set of $r + 1$ vertices in G.
Suppose for each i, $1 \leq i \leq r$, $N(x_i) = Y$.
If $\quad \forall_{1 \leq i \leq r} : w(x_i) \geq w(y_i)$ and $low \geq \max_{1 \leq i \leq r} nw(x_i)$
Then Turn Y into a clique
Remove the vertices in X and their incident edges from G.

From the assumptions in the Buddies rule, it follows that X and Y are disjoint sets. An example of the Buddies rule with $r = 3$ is given in Figure 5.

Corollary 7 *The Buddies rule is safe.*

Proof. The Buddies rule is a special case of the Contraction Reduction rule, and hence is safe by Theorem 3. The corresponding contraction reduction is the tuple $(\{x_1, \ldots, x_r\}, \{y_1, \ldots, y_{r+1}\}, \{\{x_i, y_i\} \mid 1 \leq i \leq r\})$: note that if we contract each x_i with y_i $(1 \leq i \leq r)$, then the set $\{y_1, \ldots, y_{r+1}\}$ turns into a clique. $\Box$

The Buddies rule given above generalises the Buddy rule, which is one of the rules in the set of rules to recognise graphs of treewidth three [3]. The Buddy rule is the special case of the Buddies rule with $r = 2$.

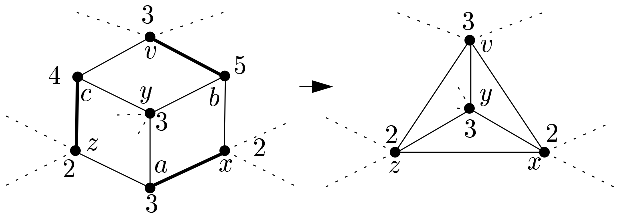

Figure 6: An instance of the *Extended Cube rule.*

The last reduction rule is taken from Arnborg and Proskurowski [3].

The Extended Cube rule
Suppose v, x, y, z, a, b, c are distinct vertices in graph $G = (V, E, w)$.
Suppose $N(a) = \{z, y, x\}$, $N(b) = \{x, y, v\}$, $N(c) = \{v, y, z\}$.
If $\quad low \geq \max(nw(a), nw(b), nw(c))$, $w(a) \geq w(x)$, $w(b) \geq w(v)$, and $w(c) \geq w(z)$
Then Turn $\{v, w, y, z\}$ into a clique
Remove the vertices in $\{a, b, c\}$ and their incident edges from G.

Corollary 8 *The Extended Cube rule is safe.*

Proof. Observe that $(\{a,b,c\},\{v,x,y,z\},\{\{z,a\},\{x,b\},\{v,c\}\})$ is a contraction reduction. Now the result follows from Theorem 3. $\square$

From the literature, we have a special case of the Extended Cube rule: the Cube rule. The Cube rule is obtained from the Extended Cube rule by additionally requiring that vertex y has degree three. In that case, y can be removed from the graph as well, (i.e., by application of the simplicial rule). The Cube rule can be implemented somewhat faster than the Extended Cube rule. We have not generalised the Extended Cube rule further, because our experiments for the unweighted case [6] indicate that this reduction rule seldom occurs. In Figure 6 an instance of the Extended Cube rule is given, which can be applied when $low \geq 90$.

All reduction rules mentioned are built upon existing rules. However, it is also possible to derive new reduction rules with help of Theorem 3. Consider Figure 7. Note that this subgraph can not be reduced using any of the rules mentioned in this section so far. Write $X = \{x_1,\ldots,x_5\}$, $Y = \{y_1,\ldots,y_5\}$. Suppose that for $1 \leq i \leq 5$, we have $w(x_i) \geq w(y_i)$. Then $(X, Y, \{\{x_i, y_i\} \mid 1 \leq i \leq 5\})$ is a contraction reduction. Hence, if for $1 \leq i \leq 5$, $low \geq nw(x_i)$, then turning Y into a clique and removing X and its incident edges is safe. In similar ways, several more rules can be designed, but it is unclear whether such rules would help much for the preprocessing of graphs that arise from practical applications. Our experiments described in Section 6 show that more complex rules (e.g., Buddies and (Extended) Cube rule) are seldom applied.

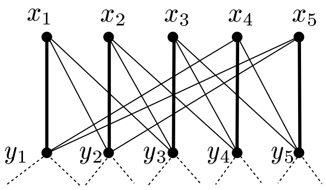

Figure 7: A contraction reduction when $w(x_i) \geq w(y_i)$ for $1 \leq i \leq 5$.

In contrast to the general contraction reduction rule, each of the rules given in this section allows a polynomial time implementation (including the time to test if a rule can be applied, and to find a possible 'spot' in the graph where the rule can be used to decrease the size of the graph.) In case of the Buddies rule, we can list for each vertex the sorted list of its neighbours, and then sort these lists lexicographically. Techniques for efficient implementations can be found e.g. in [3, 6, 14].

5 Rules to Increase the Lower Bound

When the value of the variable *low* is larger, i.e., closer to the weighted treewidth of the original graph, then the Almost Simplicial rule, Buddies rule, and Extended Cube rule, or more generally the Contraction Reduction rule, can be applied in more cases. Thus, it can be advantageous to spend time for obtaining a better lower bound on the weighted treewidth of the original graph, or of graphs obtained during preprocessing. Our experiments, reported in Section 6, support this. In this section, we discuss a method to increase the lower bound. Our lower bound heuristic is simple to implement and appears to give often good results. It builds upon Lemma 1, and the following result, which generalises a well known fact for the unweighted case.

Lemma 9 *For every graph $G = (V, E, w)$: $\tau_w(G) \geq \min_{v \in V} nw(v)$.*

Proof. Let $(\{X_i \mid i \in I\}, T = (I, F))$ be a tree decomposition of G of minimum weighted width, and a minimum size $|I|$. We claim that there is a vertex $v \in V$, and a node $i \in I$ with $v \cup N(v) \subseteq X_i$. This claim clearly holds when $|I| = 1$. Otherwise, take an arbitrary leaf node $i \in I$ from T, and suppose j is the unique neighbour of i in T. If $X_i \subseteq X_j$, then removing i from T gives a tree decomposition of G, still with minimum weighted width, but with a smaller size of $|I|$, which contradicts our assumptions. Thus, we may suppose that there is a vertex $v \in X_i$, with $v \notin X_j$. By the definition of tree decomposition, there is no $i' \neq i$ with $v \in X_{i'}$; hence, for every edge $\{v, w\}$, we have $w \in X_i$, so $v \cup N(v) \subseteq X_i$. We can conclude that there exists a vertex $v \in V$ with $\tau_w(G') \geq w(X_i) \geq nw(v)$, which proves the lemma. □

The lemma suggests the following lower bound heuristic for the weighted treewidth, called the *Maximum Minimum Neighbourhood Weight* heuristic, or MMNW for short. It repeatedly removes the vertex with the minimum neighbourhood weight until the graph is empty, and reports the maximum of these minimum neighbourhood weights seen during the process. Clearly, this number constitutes a lower bound on the weighted treewidth of the input graph since all intermediate graphs are subgraphs of G.

MMNW heuristic
Input: Graph $G = (V, E, w)$.
Output: Number ℓ with $\tau_w(G) \geq \ell$.
$\ell = 0$;
$G' = G$;
While G' is not the empty graph
Do Let v be a vertex with $nw_{G'}(v)$ minimal among all vertices in G'.
$\ell = \max\{\ell, nw_{G'}(v)\}$;
Remove v and its incident edges from G'.
Return ℓ.

A small improvement is possible to the MMNW heuristic. Instead of deleting vertices, it is also possible to contract them with a neighbour. As contracting a vertex v to a vertex x gives a vertex with weight $\min\{w(v), w(x)\}$, we refrain from contracting a vertex to a neighbour when it has only neighbours of higher weight. When the vertex v of minimum neighbourhood weight has at least one neighbour of smaller or equal weight, then we contract v to one of its neighbours; we chose the neighbour of smallest neighbourhood weight among all neighbours x with $w(x) \leq w(v)$. The resulting heuristic is called the MMNW+ heuristic.

MMNW+ heuristic
Input: Graph $G = (V, E, w)$.
Output: Number ℓ with $\tau_w(G) \geq \ell$.
$\ell = 0$;
$G' = G$;
While G' is not the empty graph
Do Let v be a vertex with $nw_{G'}(v)$ minimal among all vertices in G'.
$\ell = \max\{\ell, nw_{G'}(v)\}$;
Compute $A = \{x \mid \{v, x\} \in E \wedge w(x) \leq w(v)\}$.
If $A = \emptyset$
Then Remove v and its incident edges from G'.
Else Select the vertex x from A with $nw(x)$ minimal.
Contract the edge $\{v, x\}$ to x in G'.
(I.e., make x adjacent to all neighbours of v, and
remove v and its incident edges from G'.)
Return ℓ.

Lemma 10 *Suppose the MMNW+ heuristic, applied to G, outputs k. Then $\tau_w(G) \geq k$.*

Proof. An invariant of the algorithm is that G' is a minor of G: in each step, we either delete vertices or edges, or do an edge contraction. It follows that $\ell \leq \tau_w(G)$ is also an invariant, using Lemma 9. □

Unfortunately, it is possible that the lower bound obtained by the MMNW+ heuristic and the value of $\tau_w(G)$ differ much. Consider planar graphs with all vertices of weight two. As every planar graph has a vertex of degree at most five, the MMNW+ heuristic will never give a value larger than 2^6 on such instances, but the weighted treewidth of such planar graphs can be arbitrary large. However, in many other cases, the MMNW+ heuristic gives a reasonable lower bound for the weighted treewidth.

Other methods to increase *low* are also possible. For instance, when the graph contains a clique minor, then *low* can be set to the maximum of its current value and the weight of the clique. Such a situation would occur when a subgraph is found that fulfills the structural requirements of a contraction reduction, but not the conditions on the weights and *low*. However, lower bounds based on clique minors may be far apart from the actual treewidth (e.g., recall that planar graphs can have large treewidth but never have a clique with five vertices as a minor), and are in general intractable.

6 Computational Experiments

In this section we report on computational experiments for the described preprocessing rules. Our experiments are conducted on 23 probabilistic networks developed for real-life problems. The Alarm, BOBLO, Diabetes, Link, Munin, Oesoca, PigNet2, Pigs, VSD, and Wilson networks are taken from medical applications; several versions exist of the

Munin and Oesoca networks. The Barley and Mildew networks are used for agricultural purposes, the Water network models a water purification process and the OOW-trad, OOW-bas, OOW-solo, and Ship-ship networks are developed for maritime use. The rules are implemented in C++. All computations have been carried out on a Linux-operated PC with a 1700 MHz Intel Pentium 4 processor.

We discuss the results by means of four tables. Table 1 shows the results of preprocessing for (i) only the Simplicial rule, (ii) all described rules, and (iii) all described rules with initially *low* set to the MMNW+ lower bound. To illustrate the quality of the MMNW+ bound in comparison to the MMNW bound both values are presented in Table 2. Table 3 records the number of vertices preprocessed by each of the rules. Finally, in Table 4 the results of preprocessing for weighted treewidth are compared with those for (unweighted) treewidth.

The algorithm of Lauritzen and Spiegelhalter [13] works on a tree decomposition of the moralisation of the network. In the moralisation of a directed graph, for each pair of arcs with a common tail, an edge is added between the heads of the arc, and then all directions of arcs are dropped. Thus, in Table 1, the size of the graph after moralisation is shown (see [6] for the original sizes). Note that vertices with many incoming edges in the probabilistic network create a large clique in the moralisation. In Table 1, we show the size of the networks after each of the preprocessing strategies. Moreover the value of *low* after preprocessing is reported as well as the initial value of *low* provided by the MMNW+ lower bound. To increase readability, the last column reports the computation times for the last strategy. Table 1 shows that application of the Simplicial rule only already results in substantial graph size reductions in all cases. On average over 50% of the vertices are removed by preprocessing (with a minimum of 18% and a maximum of 87%). Including all other preprocessing rules results in even more reduced graphs. For 6 out of 23 networks the graph is preprocessed to the empty graph and hence the weighted treewidth is given by the *low* for these instances. By application of the other rules, the Simplicial rule can be applied again and induces a further increase of *low* in four cases (note that *low* is not increased by rules other than the Simplicial rule and its special cases, the Islet rule and the Twig rule).

A third reduction can be shown by the use of an initial *low* value generated by a lower bound procedure. By the MMNW+ lower bound, *low* is increased for 16 networks and for 10 out of the 17 remaining networks the graph size is reduced further. The time needed to perform preprocessing is really small in comparison to the results achieved. Even the 34 seconds for the PigNet2 network is justifiable taking into account that the graph is reduced from 3034 to 1002 vertices. We only performed computations with MMNW+ as initial lower bound. In Table 2 the MMNW and MMNW+ lower bounds are compared. Although computation times are somewhat higher, the increase of the lower bound is substantially for many instances. Most impressive example is the result for the PigNet2 network ($3^{29} \approx 6.8 * 10^{13}$). This bound indicates that it is unlikely that the algorithm of [13] can be used for this network.

Table 3 allows us to have a closer look at the effectiveness of the various rules. We analysed the most general case in which the initial *low* was set to the MMNW+ lower

instance	original		simplicial			all			MMNW+	all (with MMNW+)			CPU
	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	*low*	$\|V\|$	$\|E\|$	*low*		$\|V\|$	$\|E\|$	*low*	time (s)
Alarm	37	65	11	19	32	0	0	32	32	0	0	32	0.01
Barley	48	126	35	92	40320	29	83	40320	151200	28	81	151200	0.00
BOBLO	221	328	71	132	687820	0	0	687820	687820	0	0	687820	0.10
Diabetes	413	819	335	665	7056	332	662	7056	10608	332	662	10608	0.42
Link	724	1738	494	1349	128	339	1194	128	8192	308	1158	8192	1.88
Mildew	35	80	20	40	280000	15	31	280000	280000	15	31	280000	0.01
Munin1	189	366	108	241	600	90	220	600	250000	79	209	250000	0.07
Munin2	1003	1662	449	826	600	317	674	600	18000	241	573	18000	2.31
Munin3	1044	1745	419	790	600	174	449	2000	38400	174	449	38400	2.50
Munin4	1041	1843	436	920	600	269	720	2000	80000	269	720	80000	2.43
Munin-kgo	1066	1730	298	549	1280	95	252	2000	3000	89	246	3000	2.39
Oesoca+	67	208	30	141	1536	28	135	1536	8640	22	105	8640	0.01
Oesoca	39	67	5	7	240	0	0	240	240	0	0	240	0.01
Oesoca42	42	72	6	10	240	0	0	240	240	0	0	240	0.01
OOW-bas	27	54	19	37	5400	17	35	5400	18270	17	35	18270	0.01
OOW-solo	40	87	31	68	5400	29	66	5400	36540	29	66	36540	0.00
OOW-trad	33	72	27	59	900	25	57	900	48600	24	56	48600	0.01
PigNet2	3032	7264	1643	4556	81	1051	3835	81	3^{29}	1002	3730	3^{29}	34.35
Pigs	441	806	163	305	27	126	265	27	6561	47	134	6561	0.47
Ship-ship	50	114	39	92	600	38	91	600	144000	38	91	144000	0.00
VSD	38	62	12	21	240	0	0	240	240	0	0	240	0.00
Water	32	123	24	101	3072	22	96	3072	49152	21	94	49152	0.01
Wilson	21	27	6	8	36	0	0	108	108	0	0	108	0.00

Table 1: Preprocessing for weighted treewidth

instance	original		lower bound		CPU time (s)	
	$\|V\|$	$\|E\|$	MMNW	MMNW+	MMNW	MMNW+
Alarm	37	65	32	32	0.00	0.00
Barley	48	126	100800	201600	0.00	0.01
BOBLO	221	328	687820	687820	0.01	0.05
Diabetes	413	819	7056	34320	0.03	0.13
Link	724	1738	256	8192	0.08	0.40
Mildew	35	80	280000	280000	0.00	0.00
Munin1	189	366	1280	250000	0.01	0.04
Munin2	1003	1662	600	18000	0.15	0.67
Munin3	1044	1745	600	38400	0.16	0.71
Munin4	1041	1843	1280	80000	0.16	0.74
Munin-kgo	1066	1730	1280	3000	0.17	0.74
Oesoca+	67	208	1920	8640	0.01	0.01
Oesoca	39	67	240	240	0.00	0.00
Oesoca42	42	72	240	240	0.00	0.00
OOW-bas	27	54	18270	18270	0.00	0.01
OOW-solo	40	87	18270	36540	0.00	0.00
OOW-trad	33	72	18270	48600	0.01	0.00
PigNet2	3032	7264	243	3^{29}	1.36	5.85
Pigs	441	806	81	6561	0.04	0.15
Ship-ship	50	114	56700	144000	0.00	0.00
VSD	38	62	240	240	0.00	0.01
Water	32	123	16384	49152	0.00	0.01
Wilson	21	27	54	108	0.00	0.00

Table 2: Lower bounds for weighted treewidth

instance			number of vertices removed by rule										total
	$\|V\|$	$\|E\|$	IS	TW	SI	SE	TR	AS	BU	BS	CU	EC	
Alarm	37	70	1	11	21	1	3	0	0	0	0	0	37
Barley	48	135	0	1	13	3	2	1	0	0	0	0	20
BOBLO	221	373	1	105	80	11	24	0	0	0	0	0	221
Diabetes	413	822	0	2	76	3	0	0	0	0	0	0	81
Link	724	2257	10	73	147	12	10	164	0	0	0	0	416
Mildew	35	86	0	1	14	1	3	1	0	0	0	0	20
Munin1	189	413	0	40	42	7	21	0	0	0	0	0	110
Munin2	1003	1955	0	272	296	72	120	2	0	0	0	0	762
Munin3	1044	2066	0	298	352	74	126	8	12	0	0	0	870
Munin4	1041	2093	0	290	324	58	92	8	0	0	0	0	772
Munin-kgo	1066	1970	0	364	437	94	66	16	0	0	0	0	977
Oesoca+	67	228	0	19	18	0	0	8	0	0	0	0	45
Oesoca	39	69	1	16	21	0	1	0	0	0	0	0	39
Oesoca42	42	73	1	18	22	0	1	0	0	0	0	0	42
OOW-bas	27	57	0	1	7	1	1	0	0	0	0	0	10
OOW-solo	40	90	0	2	7	1	1	0	0	0	0	0	11
OOW-trad	33	76	0	1	5	2	1	0	0	0	0	0	9
PigNet2	3032	8311	0	71	1341	89	481	48	0	0	0	0	2030
Pigs	441	950	0	57	236	34	55	12	0	0	0	0	394
Ship-ship	50	116	0	2	9	0	1	0	0	0	0	0	12
VSD	38	71	1	16	15	2	3	1	0	0	0	0	38
Water	32	131	0	2	6	0	0	3	0	0	0	0	11
Wilson	21	29	1	12	6	2	0	0	0	0	0	0	21

IS=Islet, TW=Twig, SI=Simplicial, SE=Series, TR=Triangle, AS=Almost Simplicial, BU=Buddy, BS=Buddies, CU=Cube, EC=Extended Cube

Table 3: Contribution of the various rules

bound. The order in which the rules are applied is important for a correct interpretation of this table. The rules are applied consecutively in the order they are mentioned in the table. In case a rule applies to the current graph, the rule is executed and the search is restarted with the first rule in the order (i.e., Islet). In this way, it is avoided that, for example, a simplicial vertex is processed by the Almost Simplicial rule.

As already observed in Table 1, the majority of the vertices is preprocessed by the Simplicial rule and its specialisations Islet and Twig. Notice that Islet is only applied if singletons are detected in the graph. Hence, Link contains at least 11 components, 10 of them preprocessed completely. The Series and Triangle rules are also applied regularly, whereas the more general Almost Simplicial rule is very successful for some instances. With one exception, the remaining rules are not applied at all. Only for the munin3 network 12 vertices are removed by the Buddy rule (6 times 2 vertices).

instance	original		unweighted preprocessing		implied weighted	weighted preprocessing		implied weighted
	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	width	$\lvert V\rvert$	$\lvert E\rvert$	width
Alarm	37	65	0	0	32	0	0	32
Barley	48	126	25	76	151200	28	81	151200
BOBLO	221	328	0	0	687820	0	0	687820
Diabetes	413	819	116	276	109824	332	662	7056
Link	724	1738	308	1158	512	308	1158	512
Mildew	35	80	0	0	12681900	15	31	280000
Munin1	189	366	66	188	1512	79	209	1440
Munin2	1003	1662	165	451	7350	241	573	2401
Munin3	1044	1745	82	273	12500	174	449	2401
Munin4	1041	1843	215	642	3750	269	720	2000
Munin-kgo	1066	1730	0	0	40500	89	246	2880
Oesoca+	67	208	14	75	34560	22	105	5760
Oesoca	39	67	0	0	240	0	0	240
Oesoca42	42	72	0	0	240	0	0	240
OOW-bas	27	54	0	0	822150	17	35	5400
OOW-solo	40	87	27	63	5400	29	66	5400
OOW-trad	33	72	23	54	13500	24	56	1972
PigNet2	3032	7264	1002	3730	729	1002	3730	729
Pigs	441	806	47	134	729	47	134	729
Ship-ship	50	114	24	65	57188	38	91	600
VSD	38	62	0	0	360	0	0	240
Water	32	123	21	94	12288	21	94	12288
Wilson	21	27	0	0	108	0	0	108

Table 4: Preprocessing for unweighted and weighted treewidth

Finally, in Table 4 we compare the results for preprocessing for weighted and unweighted treewidth. For (unweighted) treewidth, the value *low* can be increased to 4 in case the Islet, Twig, Series, Triangle, Buddy, and Cube rule cannot be applied anymore (cf. [3]). Compared with preprocessing for (unweighted) treewidth, less vertices are preprocessed by the same set of rules for weighted treewidth. Hence, at first sight the conclusion from this table may be that preprocessing for weighted treewidth is less effective than preprocessing for the unweighted notion. However, this conclusion is only half the truth. Of course less vertices could be processed by the more restrictive conditions for the rules. Also the unavailability of theoretical results to increase *low* further if certain rules cannot be applied anymore limits the effect of preprocessing. However, the advantage of preprocessing for weighted treewidth lies in the fact that the maximum weight of the bags X_i is considered. By reversing the preprocessing for (unweighted) treewidth, bags can be constructed with unnecessary high $w(X_i)$. The maximum value of $w(X_i)$ by reconstruction is pointed out by the columns "implied width" in Table 4. Although we do not know the weighted treewidth in most cases, for 6 networks this value is higher than the MMNW+ lower bound indicating that the preprocessing is overreaching in this context. Most eye-catching result in this context is obtained for the VSD network. Reversing the preprocessing results in a value that is 50% higher than necessary.

7 Conclusions

Instances of NP-hard problems often can be reduced to equivalent but smaller instances using preprocessing techniques. For the problem of finding a tree decomposition for a weighted graph with minimal weighted width, we provided such a technique. Our method consists of the application of a set of reduction rules that allows for reduction of a weighted graph without increasing its weighted treewidth, and for which every reduction is easily reversible. We showed that several of the known reduction rules can be generated from the generic Contraction Reduction rule, and that new feasible reduction rules can be created. Furthermore, we presented techniques for getting lower bounds on the weighted treewidth. We leave the confluence of the rules as an interesting open problem (Conjecture 5.)

Experiments were conducted on the graphs of a set of probabilistic networks taken from real-life applications. These experiments revealed that a subset of the identified reduction rules were able to reduce the graphs significantly, or eliminate them even completely. Therefore, preprocessing by applying reduction rules is a very useful technique for the weighted treewidth problem.

Acknowledgements

We thank Linda van der Gaag for valuable suggestions and discussions for this research; furthermore we thank Kristian Kristensen, Anders L. Madsen, Kristian G. Olesen, Claus Skaaning Jensen, and Linda van der Gaag for providing instances of probabilistic networks.

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] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *J. ACM*, 40:1134–1164, 1993.

[3] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Alg. Disc. Meth.*, 7:305–314, 1986.

[4] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Disc. Appl. Math.*, 23:11–24, 1989.

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

[6] H. L. Bodlaender, A. M. C. A. Koster, F. van den Eijkhof, and L. C. van der Gaag. Pre-processing 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.

[7] H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Disc. Math.*, 6:181–188, 1993.

[8] H. L. Bodlaender and B. van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167:86–119, 2001.

[9] F. V. Jensen. *Bayesian Networks and Decision Graphs.* Statistics for Engineering and Information Science, Springer-Verlag, New York, 2001.

[10] T. Kloks. *Treewidth. Computations and Approximations.* Lecture Notes in Computer Science, Vol. 842. Springer-Verlag, Berlin, 1994.

[11] A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen. Solving partial constraint satisfaction problems with tree-decomposition. *Networks*, 40(3):170–180, 2002.

[12] J. Lagergren. The nonexistence of reduction rules giving an embedding into a k-tree. *Disc. Appl. Math.*, 54:219–223, 1994.

[13] 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.

[14] J. Matoušek and R. Thomas. Algorithms for finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.

[15] J. Pearl. *Probablistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, Palo Alto, 1988.

[16] D. P. Sanders. On linear recognition of tree-width at most four. *SIAM J. Disc. Math.*, 9(1):101–117, 1996.