Alcom-FT Technical Report Series
ALCOMFT-TR-03-124

Safe separators for treewidth

Hans L. Bodlaender

Arie M. C. A. Koster

institute of information and computing sciences, utrecht university

technical report UU-CS-2003-027

www.cs.uu.nl

Safe separators for treewidth*

Hans L. Bodlaender† Arie M. C. A. Koster‡

Abstract

A set of vertices $S \subseteq V$ is called a safe separator for treewidth, if S is a separator of G, and the treewidth of G equals the maximum of the treewidth over all connected components W of $G - S$ of the graph, obtained by making S a clique in the subgraph of G, induced by $W \cup S$. We show that such safe separators are a very powerful tool for preprocessing graphs when we want to compute their treewidth. We give several sufficient conditions for separators to be safe, allowing such separators, if existing, to be found in polynomial time. In particular, every minimal separator of size one or two is safe, every minimal separator of size three that does not split off a component with only one vertex is safe, and every minimal separator that is an almost clique is safe; an almost clique is a set of vertices W such that there is a $v \in W$ with $W - \{v\}$ a clique. We report on experiments that show significant reductions of instance sizes for graphs from probabilistic networks and frequency assignment.

1 Introduction

In several applications, it is of importance to determine the treewidth of a given graph and find a tree decompositions of graphs of minimum or close to minimum treewidth. Having such a tree decomposition allows the solution of various otherwise intractable graph problems, see, amongst many others, [3, 8, 11, 13]. Experiments and applications show that this is also useful in a practical setting. Koster et al. [18] used tree decompositions to solve instances of frequency assignment problems. The algorithm of Lauritzen and Spiegelhalter [19] to solve the probabilistic inference problem on probabilistic networks is the most commonly used algorithm for this problem and uses tree decompositions. An important problem that arises in such applications is to find a tree decomposition of the given graph of width as small as possible.

It is known that for each fixed k, there exists a linear time algorithm that checks if a given graph has treewidth at most k, and if so, finds a tree decomposition of G of width at most k [7]. Unfortunately, it appears that the constant factor of this algorithm is too big to make this algorithm usable in practice. (See [23].) So, an important task is to design practically efficient methods for finding tree decompositions of small width.

*This research was partially supported by NWO-EW and partially by EC contract IST-1999-14186: Project ALCOM-FT (Algorithms and Complexity - Future Technologies).

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

‡Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, D-14195 Berlin, Germany. koster@zib.de

In [10, 26], preprocessing methods based on graph reduction were studied. A number of 'safe reduction rules' was proposed; each such rule rewrites the graph locally, thus decreasing the number of vertices in the graph, such that a tree decomposition of optimal width for the smaller reduced graph can be easily transformed to one for the original graph. When no reductions are possible, another method must be used to solve the problem on the remaining graph. Experiments on a set of graphs, taken from probabilistic networks applications, showed that sizes of these remaining graphs were in general much smaller than the sizes of the original graphs. In some cases, reduction was sufficient for finding the optimal solution to the problem.

In this paper, we study a different form of preprocessing. Here we propose to use separators. A simple example is the following. The treewidth of a graph equals the maximum treewidth over all its biconnected components. Thus, when computing the treewidth of G, we can split G in its biconnected components, and solve every such biconnected component separately. In the terminology of this paper: separators of size one are safe, and thus we can split G on these separators. Each preprocessing step with safe separators takes a graph, and replaces it by two or more smaller graphs. This way, we obtain a collection of graphs. Solving the treewidth problem on the original instance is equivalent to solving the treewidth problem on each of the graphs in the collection. However, the graphs in the collection are usually significantly smaller than that in the original instance. We can repeat trying to find safe separators in the graphs in the collection, replacing these again by even smaller graphs, until we do not find a safe separator in any graph in the collection. Then, the treewidth of the graphs in the collection must be established by other means: this may be trivial (e.g., when the graph is complete), can be done by an exact method like branch and bound (which can be fast enough when the preprocessing yielded only small graphs in the collection), or with an approximation algorithm.

After some preliminary definitions and results in Section 2, we establish our main graph theoretic results in Section 3. In this section, we give several sufficient conditions for a separator to be safe for treewidth. In Section 4, we discuss how the safe separators can be found. In Section 5, we compare the use of safe separators with the use of graph reduction from [10, 26]. In Section 6, we discuss on experiments that we have carried out. Some final conclusions are made in Section 7.

2 Definitions and preliminary results

In this section, we give a number of definitions and a few easy or well known lemmas. We assume the reader to be familiar with standard graph terminology. In this paper, we assume graphs to be undirected and without parallel edges or self loops. For a graph $G = (V, E)$, let $n = |V|$ be the number of vertices and $m = |E|$ be the number of edges. For a vertex set $S \subseteq V$, we denote $G - S$ as the subgraph of G, induced by $V - S$, $G[V - S]$. We denote $G + \text{clique}(S)$ as the graph $(V, E \cup \{\{v, w\} \mid v, w \in S\})$.

A set of vertices $S \subseteq V$ is a *separator* in $G = (V, E)$ when $G - S$ has more than one connected component. S is a *minimal separator*, when it does not contain another separator as a proper subset. S is a *minimum separator*, when G has no separator of size smaller than S. S is a *clique separator*, when S forms a clique in G and S is a separator.

Definition 1 *A* tree decomposition *of* $G = (V, E)$ *is a pair* $(\{X_i \mid i \in I\}, T)$*, where* $\{X_i \mid i \in I\}$ *is a collection of subsets of* V *and* $T = (I, F)$ *is a tree such that:*

(i). $\bigcup_{i\in I} X_i = V$.

(ii). $\forall\{u,w\} \in E, \exists i \in I : u, w \in X_i$.

(iii). $\forall i,j,k \in I$ *: if j is on a path in T from i to k then* $X_i \cap X_k \subseteq X_j$.

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

The third condition can be equivalently stated as: for all $v \in V$, the set $I_v = \{i \in I \mid v \in X_i\}$ is connected in T.

Treewidth can also be defined with help of triangulations. A graph $G = (V,E)$ is *triangulated* if in every cycle in G with length at least four there is a chord (i.e., two non-consecutive vertices in the cycle that are adjacent). A graph $H = (V,E')$ is a *triangulation* of $G = (V,E)$ when H is triangulated and G is a subgraph of H (i.e., $E \subseteq E'$.) It is well known that the treewidth of a graph G is exactly one smaller than the minimum over all triangulations H of G of the maximum clique size of H. Moreover, when we have a triangulation H of G with maximum clique size k, we can find a tree decomposition of G of width $k-1$ in linear time (see e.g., [9]). We will use the reverse step later: given tree decomposition $(\{X_i \mid i \in I\}, T)$ of $G = (V,E)$ of width k, the graph $H = (V,E')$ with $E' = \{\{v,w\} \mid v \neq w,\ \exists i \in I : v, w \in X_i\}$ is a triangulation of G with maximum clique size $k+1$.

The following two lemmas are well known.

Lemma 2 *Let S be a separator in G. S is a minimal separator, if and only if for every component Z of $G - S$, and for every vertex $v \in S$, there is a vertex $w \in Z$ that is adjacent to v.*

Lemma 3 *Let W form a clique in graph $G = (V,E)$, and let $(\{X_i \mid i \in I\}, T = (I,F))$ be a tree decomposition of G. Then there is an $i \in I$ with $W \subseteq X_i$.*

Definition 4 *A separator S in a graph $G = (V,E)$ is* safe for treewidth*, or, in short:* safe*, when the treewidth of G equals the maximum over all components Z of $G - S$ of the treewidth of* $G[S \cup Z] + \mathrm{clique}(S)$.

The proof of the following lemma uses standard techniques (compare e.g., [2]).

Lemma 5 *For every graph G, and every separator S in G, the treewidth of G is at most the maximum over all components Z of $G - S$ of the treewidth of* $G[S \cup Z] + \mathrm{clique}(S)$.

Proof: Suppose the maximum over all components Z of $G - S$ of the treewidth of $G[S \cup Z] + \mathrm{clique}(S)$ is α. Let these components be $Z_1, \ldots, Z_r$. Then, for $1 \leq j \leq r$, we have a tree decomposition $(\{X_i^j \mid i \in I^j\}, T^j)$ of $G[S \cup Z_i] + \mathrm{clique}(S)$ with treewidth of most α. By Lemma 3, for each j, there is a node $i^j \in I^j$ with $S \subseteq X_{i^j}^j$. A tree decomposition of G can be formed by taking the disjoint union of the tree decompositions $(\{X_i^j \mid i \in I^j\}, T^j)$, $1 \leq j \leq r$, and adding one new node i^* with $X_{i^*} = S$, and making i^* adjacent to every node i^j, $1 \le j \le r$. The treewidth of this tree decomposition is at most α. $\square$

Lemma 6 uses the same idea as Lemma 5, but now formulated in terms of triangulations.

Lemma 6 *Let S be a separator in $G = (V, E)$. Suppose $W_1, \ldots, W_r$ induce the components of $G - S$, and let for each $q \in \{1, \ldots, r\}$ $H_q = (W_q \cup S, E_q)$ be a triangulation of $G[S \cup W_q] + \text{clique}(S)$, each with maximum clique size at most k. Let $H = (V, F)$ be the graph obtained from taking the (non disjoint) union of the graphs H_q, i.e., $F = \bigcup_{1 \le q \le r} E_q$. Then H is a triangulation of G with maximum clique size k.*

Proof: Let $V_q = W_q \cup S$ for all $q \in \{1, \ldots, v\}$. If v and w are adjacent, then there must be a q with $v, w \in V_q$, $1 \le q \le r$. If $v, w \in V_q$, and $v, w \in V_{q'}$, $q \ne q'$, then $v, w \in S$. It follows that for each clique Z in H, there is a H_q with Z a clique in H_q. So H has maximum clique size k.

Clearly, H contains G as a subgraph. H is triangulated: consider a cycle C with vertices $v_1, \ldots, v_s$, $s > 3$. If there is a q with $v_1, \ldots, v_s \in V_q$, then H_q and hence H contains a chord of C. Otherwise, suppose $v_i \in V_q$, $v_j \in V_{q'}$, $q \ne q'$. As S separates v_i and v_j, S contains a vertex $v_{i'}$ on the path from v_i to v_j on the cycle, and a vertex $v_{j'}$ on the other path from v_i to v_j on the cycle. As S is a clique in H, $\{v_{i'}, v_{j'}\}$ is a chord of C. $\square$

The next lemma is also nothing more than a reformulation of known results, e.g., from [22].

Lemma 7 *Let S be a separator in G that induces a clique in G. Then S is safe for treewidth.*

Proof: Note that for every component Z of $G - S$, $G[S \cup Z] + \text{clique}(S) = G[S \cup Z]$ is a subgraph of G and hence its treewidth is at most the treewidth of G. Now the lemma follows with help of Lemma 5. $\square$

Lemma 7 expresses a sufficient condition for a separator S to be safe. In the next section, we give more general sufficient conditions. By this, classes of safe separators are defined. If a graph cannot be decomposed any further by a class of safe separators, we call this decomposition *final.*

Definition 8 *S is an* almost clique*, when there is a vertex $v \in S$, such that $S - v$ forms a clique. v is called the* non-clique vertex *of almost clique S.*

S is an *almost clique separator*, when S is an almost clique and S is a separator. S is an *minimal almost clique separator*, when S is an almost clique, and S is a minimal separator.

Definition 9 *Graph $H = (V_H, E_H)$ is a* labelled minor *of $G = (V_G, E_G)$, when H can be obtained from G by a sequence of zero or more of the following operations: deletion of edges, deletion of vertices (and all adjacent edges), edge contraction that keeps the label of one endpoint: when contracting the edge $\{v, w\}$ the resulting vertex will be named either v or w.*

The following lemma is a trivial modification of the well known fact that treewidth cannot increase when taking minors.

Lemma 10 *Let H be a labelled minor of G. Then the treewidth of H is at most the treewidth of G.*

3 Conditions for safeness

In this section, we give a number of sufficient conditions for separators to be safe.

Lemma 11 *Suppose S is a separator in $G = (V, E)$. Suppose for every component Z of $G - S$, the graph $G - Z$ contains a clique on S as a labelled minor. Then S is safe for treewidth.*

Proof: By Lemma 5, it remains to show that the treewidth of G is at least the maximum over all components Z of $G - S$ of the treewidth of $G[S \cup Z] + \text{clique}(S)$, i.e., that for every component Z of $G - S$, the treewidth of $G[S \cup Z] + \text{clique}(S)$ is at most the treewidth of G. Consider a component Z. From the fact that $G - Z$ contains a clique on S as a labelled minor, it follows that G has $G[S \cup Z] + \text{clique}(S)$ as a labelled minor: when applying the operations that yield a clique on S from $G - Z$ to G, we obtain $G[S \cup Z] + \text{clique}(S)$. Thus, by Lemma 10, the treewidth of $G[S \cup Z] + \text{clique}(S)$ is at most the treewidth of G, and the lemma follows. □

Corollary 12 *Suppose S is a separator in $G = (V, E)$. Suppose for every component Z of $G - S$, the graph $G[Z \cup S]$ contains a clique on S as a labelled minor. Then S is safe for treewidth.*

Proof: Let S be a separator, and suppose that for every component Z of $G - S$, the graph $G[Z \cup S]$ contains a clique on S as a labelled minor. Consider a component Z' of $G - S$. Let Z'' be another component of $G - S$. The graph $G - Z'$ contains $G[S \cup Z'']$ as a subgraph, and hence contains a clique on S as labelled minor. The result now follows from Lemma 11. □

Theorem 13 *If S is a minimal almost clique separator of G, then S is safe for treewidth.*

Proof: Consider a minimal almost clique separator separator S in G. Let v be the non-clique vertex in S. We show that for every component Z of $G - S$, the graph $G[Z \cup S]$ contains a clique on S as a labelled minor; the lemma then follows from Corollary 12.

Consider a component Z of $G - S$, and consider the graph $G[Z \cup S]$. As v is adjacent to a vertex in Z (Lemma 2), we can contract all vertices in Z to v. The resulting graph G' has vertex set S, and is a clique: for $w, x \in S - \{v\}$, $\{w, x\}$ is an edge in G and hence in G'; for $w \in S - \{v\}$, w is adjacent to a vertex $y \in Z$ in G (Lemma 2), and hence after the contractions there is an edge $\{v, w\}$ in G'. □

In the next section, we see that there is a polynomial time algorithm to find a minimal separator that is an almost clique in a graph G when such a separator exists. Thus, Theorem 13 gives our first new method to preprocess the graph with safe separators. We also can establish safeness of some separators of small size.

Corollary 14 *Every separator of size 1 is safe for treewidth. Every minimal separator of size 2 is safe for treewidth.*

Proof: A vertex set of size 1 is a clique; a vertex set of size 2 is a clique or almost clique. □

We now consider separators of size three. We first need the following lemma.

Lemma 15 *Let S be a minimum separator of G with $|S| = 3$, and let W be the vertex set of a connected component of $G - S$ with $|W| \geq 2$. Then $G[W \cup S]$ contains a clique on S as labelled minor.*

Proof: (The following short proof of this lemma is due to Gasper Fijavz.) First, we show that $G[W \cup S]$ contains a cycle C. Suppose $G[W \cup S]$ is a forest. The vertices in W cannot have degree less than three, as G does not have separators of size one or two. Thus forest $G[W \cup S]$ has at least two vertices of degree at least three, so has at least four leaves. As only the vertices in S can be a leaf and $|S| = 3$, this is a contradiction.

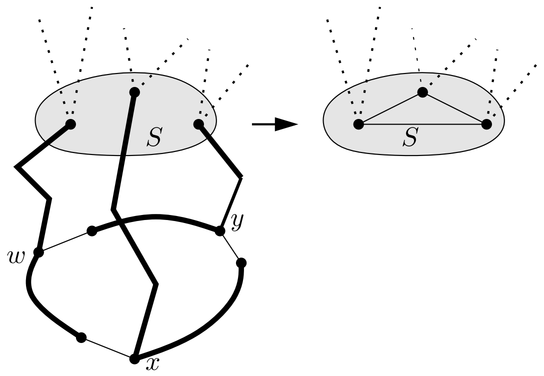

Figure 1: Illustration to the proof of Lemma 15

Now, consider cycle C, and take three arbitrary vertices w, x, y on C. As the minimum separators in G have size three, there are three vertices disjoint paths in G from w, x, y to S, by the Menger theorem [21]. As S is a separator in G, these paths belong to $G[W \cup S]$. We can now contract all edges on each of these three paths, and edges on C until we obtain a clique on S. So, a clique on S is a labelled minor of $G[W \cup S]$. See Figure 3. □

Theorem 16 *Let S be a minimum separator of size three in G. Suppose $G - S$ has two connected components, each with at least two vertices. Then S is safe for treewidth.*

Proof: The theorem directly follows from Lemma 15 and Corollary 12. □

As Theorem 16 asks for minimum separators of size three, we should first split G with help of the separators of size one and two. Once we do not have such separators left, each separator of size three is a minimum separator. Other safe separators of size three are implied by the following lemma when $k = 3$.

Lemma 17 *Let S be a minimal separator in G of size k, such that $G-S$ has at least k connected components. Then S is safe for treewidth.*

Proof: Consider a component Z of $G-S$. Let $Z_1, \ldots, Z_{k-1}$ be the components in $(G-Z)-S$, and let $S = \{v_1, \ldots, v_k\}$. By Lemma 2, for each i, $1 \leq i \leq k-1$, v_i is adjacent to a vertex in Z_i, we can contract Z_i to the vertex v_i. The result will be a clique on S, as for each i, j, $1 \leq i < j \leq k$, v_j is adjacent to a vertex in Z_i (Lemma 2), hence v_i is adjacent to v_j after the contraction of Z_i to v_i. Safeness now follows by Lemma 11. □

Together with the almost clique separators, these results are quite powerful. The only case where a separator of size three is not necessarily safe is when it is the neighbourhood of a vertex of degree three, it splits the graph into two components, and the vertices in the separator are not adjacent. For separators of size four, we can derive some cases where they are safe as well, but in this case, the result become less powerful.

Theorem 18 *Let $S = \{v, w, x, y\}$ be a separator of size four in G, with v adjacent to w, x, and y. Suppose that G has no separator of size two, and no separator of size three that contains v. If every connected component of $G-S$ has at least two vertices, then S is safe for treewidth.*

Proof: Consider $G-v$. $\{w, x, y\}$ is a separator in $G-v$, and $G-v$ has no separator of size 1 or 2. Hence, with the proof of Theorem 16, we have that for every connected component Z of $G-S = G-v-\{w,x,y\}$, a clique on $\{w,x,y\}$ is a labelled minor of $G[\{w,x,y\} \cup Z]$. As v is adjacent to w, x, and y, we hence have that a clique on S is a labelled minor of $G[Z \cup S]$. Hence, S is safe. □

4 Finding Safe separators

The results of Lemma 7, Theorem 13, Corollary 14, Theorem 16, and Theorem 18 can be used as follows. Given the input graph G, we maintain a collection of graphs, initially, this collection costs of G only. For each graph in the collection, we try to find safe separators, as indicated by these results. Below in this section, we discuss algorithms to find such safe separators. If we cannot find such a safe separator, the graph is not further processed. Otherwise, if safe separator S is found in graph G, we replace G in the collection by the graphs $G[S \cup Z] + \text{clique}(S)$, for all connected components Z of $G-S$. We repeat this until we do not find a safe separator anymore in any graph in the collection. One can observe that the treewidth of G equals the maximum treewidth of a graph in the resulting collection.

We now discuss how to find safe separators for several types of separators discussed above. Some types of separators are easy to handle. Using safe separators of size zero means splitting the graph into its connected components. The use of separators of size one corresponds to splitting the graph into its biconnected components; this can be done in $O(n+m)$ time using depth first search [24]. Splitting a graph into its triconnected components, and finding the separators of size two can also be done in linear time [15]. Clique separators are well studied and can be found in $O(nm)$ time, see [25, 20, 5, 22].

The other types of safe separators require somewhat more discussion here. We first look to the minimal separators that are an almost clique.

Lemma 19 *Suppose $G = (V, E)$ does not contain a clique separator. For every $v \in V$, $S \subseteq V - \{v\}$, S is a minimal clique separator in $G - \{v\}$, if and only if $S \cup \{v\}$ is a minimal almost clique separator in G.*

Proof: Clearly S is a clique separator in $G - \{v\}$, if and only if $S \cup \{v\}$ is an almost clique separator in G.

Suppose $W \subset S \cup \{v\}$ and $S \cup \{v\}$ are two different almost clique separators in G. If $v \in W$, then $W - \{v\}$ is a separator in $G - \{v\}$, and thus S is not a minimal separator in G. If $v \notin W$, then W is a clique, and hence G contains a clique separator, contradiction.

Suppose $X \subset S$ and S are two different clique separators in $G - \{v\}$. Then $X \cup \{v\}$ is an almost clique separator in G, and hence $S \cup \{v\}$ is not a minimal separator in G. The lemma now follows. □

The lemma tells us that we can find the set of minimal almost clique separators of $G = (V, E)$ by finding the minimal clique separators in $G - \{v\}$ for all $v \in V$.

Corollary 20 *The set of all minimal almost clique separators of a graph $G = (V, E)$ can be found in $O(n^2 m)$ time.*

Proof: For every v, we can find the minimal clique separators in $G - v$ in $O(nm)$ time [25, 20, 5, 22]. □

After we have split a graph on a minimal almost clique separator we should check the resulting graphs again for having a minimal almost clique separator. Consider the graph in Figure 2. For vertex v, there is no separator S such that $S - \{v\}$ is a clique. However, after the graph has been split on separator $\{w, x, y\}$ with $\{w, x, y\}$ turned into a clique, then the component with v contains a minimal separator $S' = \{v, w, x\}$ with $S' - \{v\}$ a clique.

We now look to algorithms to find the safe separators of size three, indicated by Theorem 16. There is an $O(n^2)$ algorithm to split a graph into its four-connected components and finding all separators of size three [16]. We conjecture that this may lead to an $O(n^2)$ time algorithm to make a decomposition of a graph with respect to safe separators of size one, of size two, and minimum separators of size three whose components each contain at least two vertices, such that each of the graphs in the resulting final decomposition does not contain such a safe separator.

In our experiments, we have used a simpler method, based on the vertex connectivity algorithm, described in [14, Section 6.2]. We also just find one safe separator, and repeat the procedure all over again on the new graphs in the collection. Using flow techniques, we can find for a pair of vertices v, w if there is a separator of size at most k that separates v from w in $O((n+m)k)$ time. This can be used to check whether there is a separator S that separates v from w of size k such that both v and w belong to a component of $G - S$ with at least two vertices in $O((n + m)k^3)$ time as follows. Of course, when v and w are adjacent, then no separator between v and w

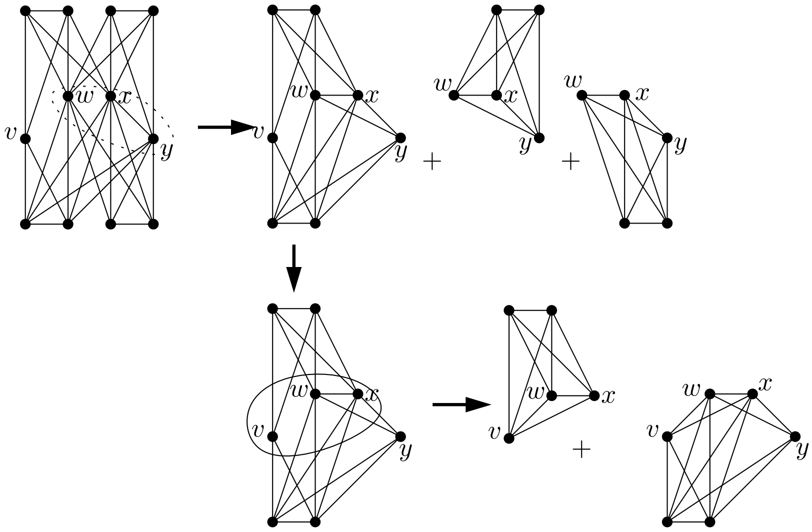

Figure 2: New minimal almost clique separators can be formed

exists. Suppose both v and w have degree at most k. Then we look to all $O(k^2)$ graphs obtained by contracting v to a neighbour, and contracting w to a neighbour, for all pairs of neighbours, excluding contractions to a vertex that is adjacent to both v and w. One can see that the required separator in G exists, if and only if there is a separator of size $\leq k$ in one of these graphs obtained by contraction: if S separates v from w in the graph obtained by contracting v to x and contracting w to y, then in G, S also separates v from w, and x belongs to the same component as v in $G - S$, and w and y also belong to the same component in $G - S$. Thus, we look to all $O(k^2)$ graphs obtained by the contractions for separators between v and w of size at most k. (Actually, when the separators are found using an application of the Ford-Fulkerson flow algorithm, we can see that it is not necessary to do the contractions to w; we omit here the technical details.) When both v and w have degree at least $k + 1$, then for every separator S of size at most k that separates v from w, both v and w have a neighbour that does not belong to S and hence belong to a component of $G - S$ of size at least two. So, in this case, we just look for a separator of size at most k between v and w. If one of v or w has degree at most k and the other has not, then we look to the $O(k)$ graphs, obtained by contracting the small degree vertex to one of its neighbours. Thus in $O((n + m)k^3)$ time, we can determine if the desired separator between v and w exists, and if so, find one. We will apply this procedure for the case that $k = 3$.

So far, we required the separator to separate a specific pair of vertices. To look for any separator in the graph, we use the scheme of [14, p. 129]. Take an arbitrary vertex v. For all vertices w, look if there is a separator of size at most k, with v and w in different components of size at least two. If we find such a separator, we are done. If not, when the degree of v is at most k, check if the neighbours of v split G into at least three components with two of them have size at least two. Otherwise, we know that v must belong to any separator of size at most k that splits G with at least two components of size at least two. Remove v from G, and look for a separator of size at most $k - 1$ in $G - v$ with at least two components of size at least two. This procedure takes $O(nmk^4)$ time; we apply it with $k = 3$, so we have an $O(nm)$ procedure to check if G has a safe separator, indicated by Theorem 16.

Minimum separators of size three that split the graph into at least three components are also safe (Lemma 17.) Most of these are already found by the procedure above; the remaining case is when there are two vertices of degree three with the same set of neighbours. We can determine in $O(n)$ time if there are two vertices of degree three with the same neighbourhood: assume some order on the vertices. Then, list all vertices of degree three with their neighbours in sorted order, and then radix sort this list (see e.g., [12, Section 9.3]). Vertices with the same neighbourhood will be on consecutive places in this list.

Thus, in $O(nm)$ time, we find for a given graph G if it has a safe separator of size three of the types, given in Theorem 16 or Lemma 17. Repeating this on newly formed graphs in the collection gives a (conservative) time bound of $O(n^2m)$ to find all safe separators of size three.

5 Safe separators and graph reduction

In [10], a set of reduction rules was established that can be used for preprocessing a graph. Generalizations of these (e.g., to the case of weighted treewidth) were considered in [26]. Several of these rules can be seen to be special cases of the use of safe separators, while for some others, there are small differences. However, it is still recommendable to use graph reduction besides safe separators, as many of the graph reduction steps can be carried out much faster than the more time consuming safe separators steps.

When applying preprocessing with the reduction rules, we maintain in a variable *low* a lower bound on the treewidth of the initial graph G. The following rules were given in [10], using results from [4] and others. For each rule, there is a method to 'reverse it', i.e., compute the treewidth and an optimal tree decomposition for the graph before the rule from those for the graph after the application of the rule.

- Simplicial rule. Remove a vertex v whose neighbours form a clique, and set *low* to the maximum of *low* and the degree of v. Special cases are when the degree of v is zero (Islet rule) or one (Twig rule).
- Almost simplicial rule. Suppose the neighbours of v form an almost clique. If *low* is at least the degree of v, then turn the neighbours of v into a clique and remove v. Special cases are when the degree of v is two (Series rule) or three (Triangle rule).
- Buddy rule. Suppose v and w have the same neighbours, and both have degree three. Suppose *low* is at least three. Then turn the neighbours of v and w into a clique and remove v and w.
- Extended cube rule. Suppose G contains the subgraph, shown in Figure 3. a, b, and c have no neighbours that are not shown. If $low \geq 3$, then turn v, w, x, and y into a clique, and remove a, b, and c.

A generalisation of the Buddy rule is the Buddies rule from [26]: suppose $v_1, \ldots, v_k$ have the same neighbours, and each of these has degree $k+1$. Then, if $low \geq k$, make the neighbours of v_1 into a clique, and remove $v_1, \ldots, v_k$.

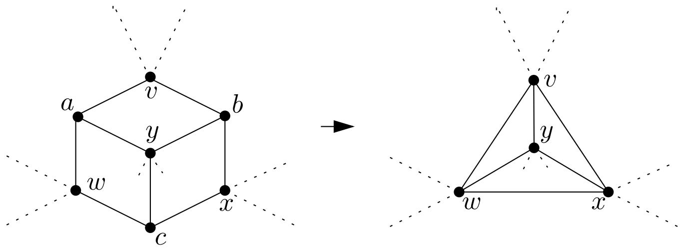

Figure 3: The Extended Cube Rule

In each of these cases, we can observe that we can use safe separators that split the graph into one or more cliques and one or more graphs that are either the same graph, obtained from applying the reduction rule or a subgraph of this graph. The technical difference is that the reduction rules check on the value of *low*, while the safe separators check on minimality of certain separating sets. There are a few mostly trivial exceptions; see also the discussion on the Extended Cube rule.

First, consider the simplicial rule, and suppose G is not a clique. The set of neighbours of v, $N(v)$ is a separator, separating v from the rest of the graph, and as it is a clique, it is safe. Thus, splitting on $N(v)$ replaces G by $G - v$ and the clique formed by $\{v\} \cup N(v)$. The case that G is a clique gives a trivial exception: in this case, no separator exists.

We now consider the buddy rule. Suppose the buddy rule can be applied in graph $G = (V, E)$, with v and w the vertices of degree three with the same set of neighbors $N(v)$. There are a few cases. The trivial case, where G contains only the five vertices in $\{v, w\} \cup N(v)$ is not very interesting. Suppose G has more than five vertices. Write $H = G - \{v, w\} - N(v)$. $N(v)$ is a separator that splits G in at least three components: one component consisting of the vertex v only, one component with vertex w only, and one or more components for each connected component of H. If $N(v)$ is a minimal separator, then it is safe, by Lemma 17, as it has size three and splits G into at least three components. In this case, using $N(v)$ as safe separator yields the following graphs: a clique on $\{v\} \cup N(v)$, a clique on $\{w\} \cup N(v)$, and for each connected component of H, the subgraph induced by the component and $N(v)$, with edges added between the vertices in $N(v)$. Note that each of these latter graphs is an induced subgraph of the graph obtained by applying the buddy rule with v and w. If $N(v)$ is not a minimal separator, then there exists a subset S of $N(v)$ that is a minimal, and hence safe separator. Using S as safe separator splits off at least one connected component of H from G. This step can be repeated with other safe separators that are a subset of $N(v)$ until the graph has only five vertices or $N(v)$ is a minimal (hence safe) separator. We can observe that every graph yielded by the procedure is either a clique, or an induced subgraph of the graph obtained by applying the buddy rule with v and w to G.

The analysis for the Almost Simplicial rule is similar. When v is almost simplicial, then $N(v)$ is an almost clique. Now, we can always split on safe separators that are $N(v)$ or a subset of $N(v)$, and this yields a number of graphs, each either a clique, an induced subgraph of the graph obtained by applying the rule to almost simplicial vertex v, or a graph with at most four vertices.

Finally, we consider the Extended Cube rule. Suppose the rule can be carried out in G. We name the participating vertices as in Figure 3. $\{v, w, x, y\}$ is a separator. A somewhat tedious case analysis shows that $\{v, w, x, y\}$ is a safe separator or contains a safe separator as subset, allowing to repeat splitting of graphs until/unless we are left with a graph containing at most the seven vertices in $\{a, b, c, v, w, x, y\}$. However, in some cases, safeness is implied by Lemma 11 but not by any of the later given lemmas, and hence the separator is not found by our experiments. This happens e.g., when $G - \{a, b, c, v, w, x, y\}$ is connected. Our experiments however showed that graphs where the Extended Cube rule can be applied hardly appear in practice [10].

While most of the power of the reduction rules is also captured by the safe separators, the reduction rules have the advantage that it is much faster to find vertices in the graph where they can be applied. Thus, for a faster algorithm, one would first reduce the graph with help of the reduction rules, and when such reductions cannot be found anymore, then one can try to split the graph with help of safe separators.

6 Experiments

The study of safe separators has been initiated by our research series to efficient algorithms for determining the treewidth of graphs [10, 17, 26]. Therefore, in this section we not only discuss the direct impact of safe separators on graphs from various applications, but also their impact on the performance of other methods (e.g., heuristics) for computing treewidth. In Section 6.1, we discuss the results for applying different sets of safe separators, whereas Section 6.2 is devoted to a comparison of some heuristics before and after safe separator decomposition.

For all algorithms it holds that they have been implemented in C++, and that the computations have been carried out on a Linux-operated PC with a 2.53 GHz Intel Pentium 4 processor. For our computational studies, we used two sets of graph instances. The first set consists of the moralised graphs for probabilistic networks. Probabilistic networks are used as underlying technology in several decision support systems; the networks come from medical, agricultural, and other applications. The second set is taken from the CALMA project on frequency assignment problems [1]. In total, 15 graphs from probabilistic networks and 25 from frequency assignment are considered.

6.1 Finding safe separators

Before applying the safe separators the graphs are preprocessed by the reduction rules presented in [10]. As pointed out in Section 5, this reduces the time for the preprocessing, and we avoid the detection of *trivial separators*, i.e., separators that can also be interpreted as one of the preprocessing rules. Such separators generate lots of small graphs that can be neglected anyway.

We carried out three experiments on these instances: the effect of (i) clique separators, (ii) clique and almost clique separators, and (iii) clique and almost clique separators as well as separators of size three. For the first experiment, Table 1 shows the results for those graphs that contain clique separators. The graphs not reported on do not contain clique separators. This in particularly holds for the 'graph*' frequency assignment instances. These instances are generated according to some criteria and seem not to contain clique separators. The column

instance	size		cs	Output				Sizes of output graphs	CPU
	$\|V\|$	$\|E\|$		$\#G$	*low*	clq	todo	#vertices (# graphs)	time
munin2-pp	167	455	6	7	4	0	7	95(1), 18(2), 17(2), 8(2)	0.05
munin3-pp	96	313	2	3	4	0	3	82(1), 9(2)	0.02
munin4-pp	217	646	2	3	4	0	3	177(1), 23(2)	0.07
munin-kgo-pp	16	41	1	2	5	0	2	9(2)	0.00
celar01-pp	157	804	1	2	6	0	2	110(1), 47(1)	0.03
celar03-pp	81	413	5	6	8	3	2	63(1), 10(1), 7(1), 5(1), 4(1), 2(1)	0.03
celar07-pp	92	521	3	4	9	1	2	71(1), 16(1), 8(1), 3(1)	0.03
celar08-pp	189	1016	3	4	9	0	3	120(1), 53(1), 16(1), 8(1)	0.07
celar09-pp	133	646	1	2	9	0	2	120(1), 16(1)	0.05
celar10-pp	133	646	1	2	9	0	2	120(1), 16(1)	0.05
celar11-pp	96	470	1	2	7	0	2	80(1), 19(1)	0.03

Table 1: The effect of clique separators

'cs' gives the number of clique separators found. The column $\#G$ denotes the number of output graphs. The column 'clq' tells how many of the output graphs are cliques (and hence, the treewidth of these is trivial to determine; the column 'todo' gives the number of output graphs that are not a clique and whose size is larger than the lower bound *low*, and hence their treewidth must still be determined in some way. In *low*, the lower bound on the treewidth obtained is given. This value is the maximum of (i) the lower bound provided by the graph reduction rules, (ii) the sizes of the (minimal almost) clique separators, and (iii) the sizes of the output graphs that are cliques. The column 'Sizes of output graphs' reports on the number of vertices in the graphs obtained by the decomposition and the number of graphs of that size. For example, the graph munin3-pp is decomposed in one graph of 82 vertices and 2 of 9 vertices. Note that by each decomposition the total number of vertices (over all graphs) increases by at least one. Table 1 shows that sometimes large components are decomposed from the rest of the graph; by that reducing the size of the largest component significantly. The computation times (in seconds) for this separation are very small.

In our second experiment we added the almost clique separators. Table 2 reports on the results achieved by the application of both safe separators. The column 'acs' reports the number of almost clique separators found. Again, instances that are not decomposed are left out.

In our final experiment we included the safe separators of size three as well. Table 3 shows the results in this case. The column 's3' gives the number of safe separators of size three that are found. This time we report on all instances, regardless whether the separators decompose the graph or not, in order to provide an impression on the computation times for those instances where no decomposition is achieved.

Table 3 shows that both the almost-clique separators and safe separators of size three are effective in preprocessing the graph. In particular, the minimal almost clique separators turned out to be very effective. For instance celar01-pp and celar07-pp an additional clique separator could be found after application of the almost-clique separators as described in Section 4. Separators of size 3 are found rarely but in some cases they indeed exist. Very succesfull applications of safe separators are for instance munin2-pp, munin4-pp, celar01-pp, and celar04-pp. On the other hand, the tested safe separators did not have any effect on the instances celar02-pp, graph14-pp, oesoca+-pp, ship-ship-pp, and pignet2-pp. In particular for pignet2-pp this is a pity since this graph is the largest of all test instances. For this graph and several 'graph*' instances the

instance	size		Separators		Output				Sizes of output graphs	CPU
	$\|V\|$	$\|E\|$	cs	acs	$\#G$	*low*	clq	todo	#vertices (# graphs)	time
barley-pp	26	78	0	7	8	5	7	1	16(1), 6(3), 5(3), 4(1)	0.05
diabetes-pp	116	276	0	85	86	4	84	2	8(1), 6(1), 5(84)	5.09
munin1-pp	66	188	0	2	3	4	2	1	63(1), 5(2)	0.39
munin2-pp	167	455	6	13	20	4	8	12	18(4), 17(4), 16(2), 8(2), 5(6), 4(2)	0.33
munin3-pp	96	313	2	2	5	4	2	3	79(1), 9(2), 5(2)	0.49
munin4-pp	217	646	3	4	8	4	2	6	55(2), 38(2), 23(2), 5(2)	0.70
munin-kgo-pp	16	41	1	0	2	5	0	2	9(2)	0.01
pathfinder-pp	12	43	0	5	6	6	6	0	7(5), 6(1)	0.01
pigs-pp	48	137	0	1	2	5	1	1	47(1), 6(1)	0.21
water-pp	22	96	0	1	2	6	1	1	21(1), 7(1)	0.04
celar01-pp	157	804	2	19	22	8	18	3	61(1), 47(1), 19(1), 9(1), 8(3), 7(5), 6(1), 5(6), 4(3)	4.38
celar03-pp	81	413	5	17	23	10	21	1	38(1), 11(1), 10(2), 9(2), 8(1), 7(1), 6(2), 5(7), 4(4), 3(1), 2(1)	0.54
celar04-pp	114	524	0	22	23	8	19	2	65(1), 16(1), 9(2), 8(3), 7(1), 6(3), 5(7), 4(4), 3(1)	5.75
celar05-pp	80	426	0	13	14	8	12	2	47(1), 19(1), 9(1), 8(1), 6(2), 5(4), 4(4)	0.84
celar06-pp	16	101	0	1	2	11	2	0	12(2)	0.01
celar07-pp	92	521	4	12	17	11	14	2	45(1), 16(1), 12(2), 7(2), 6(6), 5(2), 4(1), 3(2)	0.78
celar08-pp	189	1016	4	28	33	11	28	3	82(1), 39(1), 16(1), 12(3), 10(2), 8(2), 7(1), 6(2), 5(15), 4(5)	3.72
celar09-pp	133	646	2	20	23	11	21	2	82(1), 16(1), 12(2), 6(1), 5(13), 4(5)	3.08
celar10-pp	133	646	2	20	23	11	21	2	82(1), 16(1), 12(2), 6(1), 5(13), 4(5)	3.07
celar11-pp	96	470	1	8	10	7	7	3	61(1), 19(1), 13(1), 5(3), 4(4)	2.58
graph01-pp	89	332	0	4	5	9	4	1	85(1), 10(4)	4.95
graph02-pp	179	659	0	3	4	7	3	1	176(1), 8(3)	68.83
graph03-pp	79	293	0	8	9	6	8	1	71(1), 7(8)	4.94
graph04-pp	179	678	0	6	7	7	6	1	173(1), 8(6)	89.38
graph05-pp	91	394	0	4	5	9	4	1	87(1), 10(4)	5.49
graph06-pp	180	790	0	3	4	9	3	1	177(1), 10(3)	58.68
graph07-pp	180	790	0	3	4	9	3	1	177(1), 10(3)	52.82
graph08-pp	314	1173	0	18	19	8	18	1	296(1), 9(18)	1968.30
graph09-pp	405	1525	0	7	8	9	7	1	398(1), 10(7)	1932.64
graph10-pp	328	1253	0	22	23	7	22	1	306(1), 8(11), 7(5), 6(6)	1854.44
graph11-pp	307	1338	0	18	19	8	18	1	289(1), 9(18)	1912.87
graph12-pp	312	1177	0	64	65	6	64	1	248(1), 7(64)	3697.39
graph13-pp	420	1772	0	44	45	7	44	1	376(1), 8(44)	12209.73

Table 2: The effect of clique and almost-clique separators

instance	Size		Separators			Output				Sizes of output graphs	CPU
	$\|V\|$	$\|E\|$	cs	acs	s3	$\#G$	*low*	clq	todo	#vertices (# graphs)	time
barley-pp	26	78	0	7	0	8	5	7	1	16(1), 6(3), 5(3), 4(1)	0.06
diabetes-pp	116	276	0	85	0	86	4	84	2	8(1), 6(1), 5(84)	5.37
link-pp	308	1158	0	0	0	1	4	0	1		36.60
munin1-pp	66	188	0	2	0	3	4	2	1	63(1), 5(2)	0.80
munin2-pp	167	455	6	13	4	24	4	12	12	18(2), 17(4), 16(4), 6(2), 5(10), 4(2)	0.54
munin3-pp	96	313	2	2	2	7	4	4	3	79(1), 7(2), 5(4)	1.17
munin4-pp	217	646	3	4	0	8	4	2	6	55(2), 38(2), 23(2), 5(2)	1.65
munin-kgo-pp	16	41	1	0	2	4	5	2	2	7(2), 5(2)	0.01
oesoca+-pp	14	75	0	0	0	1	9	0	1		0.01
oow-trad-pp	23	54	0	0	1	2	4	1	1	21(1), 5(1)	0.07
oow-solo-pp	27	63	0	0	1	2	4	0	2	16(1), 14(1)	0.07
pathfinder-pp	12	43	0	5	0	6	6	6	0	7(5), 6(1)	0.01
pignet2-pp	1024	3774	0	0	0	1	4	0	1		3824.53
pigs-pp	48	137	0	1	0	2	5	1	1	47(1), 6(1)	0.39
ship-ship-pp	30	77	0	0	2	3	4	0	3	24(1), 6(2)	0.18
water-pp	22	96	0	1	0	2	6	1	1	21(1), 7(1)	0.08
celar01-pp	157	804	2	19	1	23	8	18	3	58(1), 47(1), 19(1), 9(1), 8(3), 7(5), 6(2), 5(6), 4(3)	5.84
celar02-pp	19	115	0	0	0	1	6	0	1		0.03
celar03-pp	81	413	5	17	0	23	10	21	1	38(1), 11(1), 10(2), 9(2), 8(1), 7(1), 6(2), 5(7), 4(4), 3(1), 2(1)	0.74
celar04-pp	114	524	0	22	1	24	8	19	2	62(1), 16(1), 9(2), 8(3), 7(1), 6(4), 5(7), 4(4), 3(1)	6.92
celar05-pp	80	426	0	13	0	14	8	12	2	47(1), 19(1), 9(1), 8(1), 6(2), 5(4), 4(4)	1.18
celar06-pp	16	101	0	1	0	2	11	2	0	12(2)	0.01
celar07-pp	92	521	4	12	0	17	11	14	2	45(1), 16(1), 12(2), 7(2), 6(6), 5(2), 4(1), 3(2)	1.10
celar08-pp	189	1016	4	32	1	38	11	33	3	76(1), 39(1), 16(1), 12(3), 10(2), 8(2), 7(1), 6(2), 5(19), 4(6)	6.02
celar09-pp	133	646	2	24	1	28	11	26	2	76(1), 16(1), 12(2), 6(1), 5(17), 4(6)	5.22
celar10-pp	133	646	2	24	1	28	11	26	2	76(1), 16(1), 12(2), 6(1), 5(17), 4(6)	5.22
celar11-pp	96	470	1	8	1	11	7	7	4	48(1), 19(1), 16(1), 13(1), 5(3), 4(4)	3.11
graph01-pp	89	332	0	4	0	5	9	4	1	85(1), 10(4)	5.68
graph02-pp	179	659	0	3	0	4	7	3	1	176(1), 8(3)	71.63
graph03-pp	79	293	0	8	0	9	6	8	1	71(1), 7(8)	5.63
graph04-pp	179	678	0	6	0	7	7	6	1	173(1), 8(6)	92.53
graph05-pp	91	394	0	4	0	5	9	4	1	87(1), 10(4)	6.56
graph06-pp	180	790	0	3	0	4	9	3	1	177(1), 10(3)	63.62
graph07-pp	180	790	0	3	0	4	9	3	1	177(1), 10(3)	63.54
graph08-pp	314	1173	0	18	0	19	8	18	1	296(1), 9(18)	1960.48
graph09-pp	405	1525	0	7	0	8	9	7	1	398(1), 10(7)	1942.06
graph10-pp	328	1253	0	22	0	23	7	22	1	306(1), 8(11), 7(5), 6(6)	1859.36
graph11-pp	307	1338	0	18	0	19	8	18	1	289(1), 9(18)	1896.00
graph12-pp	312	1177	0	64	0	65	6	64	1	248(1), 7(64)	3619.29
graph13-pp	420	1772	0	44	0	45	7	44	1	376(1), 8(44)	12381.94
graph14-pp	395	1325	0	0	0	1	4	0	1		410.75

Table 3: The effect of clique, almost clique, and size three separators for preprocessed frequency assignment instances

computations times are quite large, mainly due to the almost-clique separators that have to be repeated for all newly constructed graphs. In some cases, like barley-pp and munin2-pp, the sizes of the output graphs are small enough to expect that finding an exact solution with branch and bound can be done efficiently.

6.2 Heuristics and safe separator decompositions

For the graphs that cannot be split anymore with help of the studied safe separators, other methods have to be applied to find the treewidth. These can be exact methods, like branch and bound, or heuristics. In [17, revised version], we have compared several heuristics for approximating treewidth. The most promising are based on the triangulation algorithms minimum fill-in (MINFIL, MINFIL+MC), minimum degree (MINDEG, MINDEG+MC), Maximum Cardinality Search (MCS, MCS+MC), and Maximal Cardinal Search Minimal (MCS-M). Here, the annex +MC denotes that a chordal minimization algorithm [6] is run afterwards.

In Table 4 we compare the values and computation times of the MCS-M heuristic for the original graphs, the graphs preprocessed by the graph reduction rules, and the graphs decomposed by safe separators. Moreover, we report on the lower bound provided by the graph reduction rules [10] and the one that results from the safe separator decomposition, as already listed in the previous tables.

The results show that an additional significant time reduction can be achieved by the safe separators for most instances. Only for the 'graph*' the reductions are marginal since the size of the largest graph is reduced only marginal, cf. 3. In addition, better widths and better lower bounds are derived occasionally. Most remarkable in this context is the instance diabetes, where the width is reduced from 35 via 20 to 4, the treewidth for this instance.

We have carried out a similar experiment for the MINFIL, MINFIL+MC, MINDEG, and MINDEG+MC heuristics. As these heuristics are much faster than the MCS-M heuristic, the savings in the time for applying the heuristic were often less than the time needed for the preprocessing with safe separators. However, also here there were several cases where a reduction on the treewidth was obtained with help of safe separators.

7 Conclusions

In this paper, we introduced the notion of separators that are safe for treewidth. It was known that clique separators are safe, in our terminology. We have established a number of sufficient conditions for separators to be safe. Experiments show that such safe separators can be efficiently found, and help to reduce the problem size when we want to compute the treewidth and optimal tree decompositions for many graphs coming from practical applications. In many cases, safe separators help to reduce the instances to sizes that are small enough to make it feasible to solve the treewidth problem approximately or even exactly with a method like branch and bound. Thus, safe separators are a useful tool when preprocessing graphs for treewidth.

In an earlier paper, graph reduction was used for preprocessing [10], see also [26]. A comparison shows that most reduction rules can be obtained as a special case of applying safe separators.

instance	Original		with Graph Red.			with Safe Sep.		
	width	CPU time	*low*	width	CPU time	*low*	width	CPU time
barley	7	0.16	4	7	0.04	5	7	0.01
diabetes	35	182.77	4	20	5.00	4	4	0.00
link	37	857.43	4	36	117.83	4	36	117.83
munin1	15	8.27	4	13	0.89	4	13	0.83
munin2	16	444.31	4	8	4.29	4	7	0.13
munin3	15	662.03	4	13	2.49	4	13	1.60
munin4	28	431.79	4	15	12.05	4	11	1.13
munin-kgo	13	335.74	5	5	0.01	5	5	0.00
oesoca+	11	0.31	9	11	0.01	9	11	0.01
oow-trad	6	0.07	4	6	0.04	4	6	0.03
oow-solo	6	0.12	4	6	0.05	4	6	0.02
pathfinder	6	0.43	5	6	0.00	6	-	-
pignet2	239	77479.39	4	230	9758.43	4	230	9758.43
pigs	18	50.88	4	11	0.28	5	11	0.26
ship-ship	9	0.18	4	9	0.08	4	9	0.04
water	10	0.06	5	10	0.04	6	10	0.03
celar01	17	71.73	6	17	7.31	8	16	1.12
celar02	10	1.00	6	10	0.01	6	10	0.01
celar03	16	9.11	8	16	1.15	10	16	0.20
celar04	16	41.65	6	16	4.04	8	16	0.88
celar05	15	8.78	6	15	1.44	8	16	0.40
celar06	11	1.16	9	11	0.01	11	-	-
celar07	18	10.21	9	18	1.91	11	18	0.37
celar08	19	81.76	9	19	11.31	11	18	1.76
celar09	18	48.23	9	19	6.34	11	18	1.55
celar10	18	49.35	9	19	6.34	11	18	1.55
celar11	16	40.29	7	16	2.26	7	16	0.45
graph01	27	4.12	8	27	3.17	9	27	2.84
graph02	57	43.76	6	55	34.92	7	56	34.12
graph03	24	3.72	5	25	2.26	6	23	1.66
graph04	61	45.17	6	59	35.94	7	58	33.45
graph05	28	4.42	8	29	3.60	9	30	3.21
graph06	60	47.06	8	58	37.72	9	58	36.60
graph07	60	47.21	8	58	37.71	9	58	36.90
graph08	104	276.65	7	103	234.95	8	103	203.66
graph09	128	706.53	8	128	551.76	9	125	530.38
graph10	105	292.78	4	105	274.94	7	105	231.02
graph11	106	295.77	7	104	239.45	8	104	206.22
graph12	99	267.72	5	99	227.94	6	94	131.22
graph13	146	817.39	6	143	681.38	7	140	523.76
graph14	145	770.40	4	139	547.31	4	139	547.31

Table 4: Maximum Cardinality Search-Minimal for instances

Safe separators thus are a more powerful tool, as there are many graphs that cannot be reduced with the reduction rules, but contain safe separators. However, the algorithms for applying graph reduction are much faster than those for finding safe separators, and thus the best practice seems to be to first apply graph reduction until this is not possible, and then look for safe separators in the graph.

An open problem is to obtain faster algorithms to find the safe separators, especially the minimal almost clique separators and/or safe separators of size three (or four), and to find safe separator decompositions that are final for the given types of safe separators.

References

[1] K. I. Aardal, C. A. J. Hurkens, J. K. Lenstra, and S. R. Tiourine. Algorithms for radio link frequency assignment: The CALMA project. *Operations Research*, 50(6):968 – 980, 2003.

[2] 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.

[3] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340, 1991.

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

[5] A. Bery and J.-P. Bordat. Decomposition by clique minimal separators. Research report, LIM, Marseiller, 1997.

[6] J. R. S. Blair, P. Heggernes, and J. Telle. A practical algorithm for making filled graphs minimal. *Theor. Comp. Sc.*, 250:125–141, 2001.

[7] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[8] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In I. Privara and P. Ruzicka, editors, *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS'97, Lecture Notes in Computer Science, volume 1295*, pages 19–36, Berlin, 1997. Springer-Verlag.

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

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

[11] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Mass., USA, 1989.

[13] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comp. Sc.*, 109:49–82, 1993.

[14] S. Even. *Graph Algorithms*. Pitman, London, 1979.

[15] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2:135–158, 1973.

[16] A. Kanevsky and V. Ramachandran. Improved algorithms for graph four-connectivity. *J. Comp. Syst. Sc.*, 42:288–306, 1991.

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

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

[19] 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.

[20] H.-G. Leimer. Optimal decomposition by clique separators. *Disc. Math.*, 113:99–123, 1993.

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

[22] K. G. Olesen and A. L. Madsen. Maximal prime subgraph decomposition of Bayesian networks. Technical report, Department of Computer Science, Aalborg University, Aalborg, Denmark, 1999.

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

[24] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[25] R. E. Tarjan. Decomposition by clique separators. *Disc. Math.*, 55:221–232, 1985.

[26] F. van den Eijkhof, H. L. Bodlaender, and A. M. C. A. Koster. Safe reduction rules for weighted treewidth. ZIB-report 02–49, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany, 2002.