Available online at www.sciencedirect.com
SCIENCE DIRECT
Discrete Applied Mathematics 145 (2005) 143–154
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Tree decompositions with small cost☆

Hans L. Bodlaender[a], Fedor V. Fomin[b]

[a] *Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands*
[b] *Department of Informatics, University of Bergen, Norway*

Received 2 February 2002; received in revised form 16 August 2002; accepted 16 January 2004
Available online 25 September 2004

Abstract

The *f*-cost of a tree decomposition $(\{X_i \mid i \in I\},\ T=(I,F))$ for a function $f:\mathbf{N}\to\mathbf{R}^+$ is defined as $\sum_{i\in I} f(|X_i|)$. This measure associates with the running time or memory use of some algorithms that use the tree decomposition. In this paper, we investigate the problem to find tree decompositions of minimum *f*-cost. A function $f:\mathbf{N}\to\mathbf{R}^+$ is fast, if for every $i\in\mathbf{N}$: $f(i+1)\geqslant 2\,f(i)$. We show that for fast functions *f*, every graph *G* has a tree decomposition of minimum *f*-cost that corresponds to a minimal triangulation of *G*; if *f* is not fast, this does not hold. We give polynomial time algorithms for the problem, assuming *f* is a fast function, for graphs that have a polynomial number of minimal separators, for graphs of treewidth at most two, and for cographs, and show that the problem is NP-hard for bipartite graphs and for cobipartite graphs. We also discuss results for a weighted variant of the problem derived of an application from probabilistic networks.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Treecost; Treewidth; Minimal separator; Triangulation; Probabilistic network

1. Introduction

It is well known that many problems that are intractable on general graphs become linear or polynomial time solvable on graphs of bounded treewidth. These algorithms often have the following form: first a tree decomposition of small treewidth is made, and then a dynamic programming algorithm is used, computing a table for each node of the tree. The time to process one node of the tree is exponential in the size of the associated set of vertices of the graph; thus, when the maximum size of such a set is bounded by a constant (i.e., the width of the tree decomposition is bounded by a constant), then the algorithm runs in linear time. However, two different tree decompositions of the same graph with the same width may still give different running times, e.g., when one has many large vertex sets associated to nodes, while the other has only few large vertex sets associated to nodes.

In several applications, the same tree decomposition will be used for several successive runs of an algorithm, e.g., with different data. An important example of such an application is the PROBABILISTIC INFERENCE problem on probabilistic networks. (This application will be briefly discussed in Section 8.) Hence, in many cases it makes sense to do more work on finding a good tree decomposition, and to use a more refined measure on what is a 'good' tree decomposition. Apart from extensive studies on the problem on the notion of treewidth and the notion of 'fill-in', more precise measures have been studied mainly in the context of probabilistic networks (see [22].)

E-mail addresses: hansb@cs.uu.nl (H.L. Bodlaender), fomin@ii.uib.no (F.V. Fomin).
☆ This research was partially supported by EC Contract IST-1999-14186: Project ALCOM-FT (Algorithms and Complexity - Future Technologies).

0166-218X/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2004.01.008

In this paper, we study a notion that more closely reflects the time needed when using the tree decomposition. Suppose processing a node of the tree decomposition whose associated set has size k costs $f(k)$ of some resource (e.g., time or space). Then, processing a tree decomposition of the form $(\{X_i \mid i \in I\}, T = (I, F))$ costs $\sum_{i \in I} f(|X_i|)$. (For precise definitions, see Section 2.) We call this measure the *f-cost* of the tree decomposition; the treecost of a graph G with respect to f is the minimum f-cost of a tree decomposition of G. In this paper, we investigate the problem of finding tree decompositions of minimum f-cost. In Section 10 we discuss in more detail how far this notion comes close to precisely measuring the resources needed by the algorithm.

It appears that it is important whether the function f satisfies a certain condition which we call *fast*: a function $f : \mathbf{N} \to \mathbf{R}^+$ is fast, if for every k, $f(k+1) \geqslant 2f(k)$. Most applications of treewidth in our framework will have functions that are fast (in particular, many of the classical algorithms using tree decompositions for well-known graph problems have fast cost functions.) To a tree decomposition we can associate a triangulation (chordal supergraph) of input graph G in a natural way. Now, every graph has a tree decomposition of minimum f-cost that can be associated with a *minimal* triangulation, if and only if f is fast. This will be shown in Section 3. This result will be used in later sections to show that the problem of finding minimum f-cost tree decompositions can be solved in polynomial time for graphs that have a polynomial number of separators (Section 4), and in linear time for cographs (Section 5), and for graphs of treewidth at most two (Section 6); assuming in each case that f is fast and polynomial time computable. In Section 7, we discuss a conjecture on the relation between triangulations of minimum f-cost and minimum treewidth, and show that for a fixed k, one can find a triangulation of minimum f-cost among those of treewidth at most k in polynomial time. A variant of the problems for weighted graphs with an application to probabilistic networks is discussed in Section 8. In Section 9, we show the unsurprising but unfortunate result that for each fast f, the TREECOST$_f$ problem is NP-hard for cobipartite graphs and for bipartite graphs. Also, in these cases there is no constant factor approximation algorithm, unless $P = \text{NP}$. Some final remarks are made in Section 10.

2. Preliminaries

We use the following notations: $G = (V, E)$ is an undirected and finite graph with vertex set V and the edge set E, assumed to be without self-loops or parallel edges. Unless otherwise specified, n denotes the number of vertices and m the number of edges of G. The *(open) neighborhood* of a vertex v in a graph G is $N_G(v) = \{u \in V : \{u, v\} \in E\}$ and the *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. For a vertex set $S \subseteq V$ we denote $N_G[S] = \bigcup_{v \in S} N[v]$ and $N(S) = N[S] \backslash S$. If G is clear from the context, we write $N(v)$, $N[v]$, etc. $d_G(v) := |N_G(v)|$ is the degree of v in G. $G - v$ is the graph, obtained by removing v and its incident edges from G.

For a set $S \subseteq V$ of vertices of a graph $G = (V, E)$ we denote by $G[S]$ the subgraph of G induced by S. A set $W \subseteq V$ of vertices is a *clique* in graph $G = (V, E)$ if $G[W]$ is a complete graph, i.e. every pair of vertices from W induces an edge of G. A set $W \subseteq V$ of vertices is a *maximal clique* in $G = (V, E)$, if W is a clique in G and W is not a proper subset of another clique in G.

A *chord* of a cycle C is an edge not in C that has both endpoints in C. A *chordless cycle* in G is a cycle of length more than three that has no chord. A graph G is *chordal* if it does not contain a chordless cycle.

A *triangulation* of a graph G is a graph H on the same vertex set as G that contains all edges of G and is chordal. A *minimal* triangulation of G is a triangulation H such that no proper subgraph of H is a triangulation of G.

Definition. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, with $\{X_i \mid i \in I\}$ a family of subsets of V and T a tree, such that

- $\bigcup_{i \in I} X_i = V$.
- For all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$.
- For all $i_0, i_1, i_2 \in I$: if i_1 is on the path from i_0 to i_2 in T, then $X_{i_0} \cap X_{i_2} \subseteq X_{i_1}$.

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

The following well-known result is due to Gavril [12].

Theorem 1 (*Gavril [12]*). *Graph G is chordal if and only there is a* clique tree *of G, i.e. tree decomposition* $(\{X_i \mid i \in I\}, T = (I, F))$ *of G such that for every node i of T there is a maximal clique W of G such that* $X_i = W$.

A vertex $v \in V$ is *simplicial* in graph $G = (V, E)$, if $N_G(v)$ is a clique. Every chordal graph on at least two vertices contains at least two simplicial vertices [11].

Definition. For a function $f : \mathbf{N} \to \mathbf{R}^+$, the *f-cost* of a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\sum_{i \in I} f(|X_i|)$. The *treecost* with respect to *f* of a graph *G* is the minimum *f*-cost of a tree decomposition of *G*, and is denoted $\mathrm{tc}_f(G)$.

Definition. The *f-cost* of a chordal graph *G* is

$$\mathrm{cost}_f(G) = \sum_{W \subseteq V;\, W \text{ is a maximal clique}} f(|W|).$$

We identify the following computational problem. Given a function $f : \mathbf{N} \to \mathbf{R}^+$, the $\textsc{Treecost}_f$ problem is the problem, that given a graph $G = (V, E)$ and an integer *K*, decides whether $\mathrm{tc}_f(G) \leqslant K$.

There is a one-to-one correspondence between chordal supergraphs *H* of a graph *G* and tree decompositions of *G*: to tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ we associate the graph *H* with $E_H = \{\{v, w\} \mid \exists i \in I : v, w \in X_i\}$, and given a chordal graph *H*, one can build a tree decomposition with each set X_i a maximal clique in *H*. See e.g. [4, Section 6]. The *f*-costs of this tree decomposition and chordal supergraph are the same, hence we have:

Lemma 2. *The treecost of a graph G with respect to f equals the minimum f-cost of a chordal graph H that contains G as a subgraph*.

An interesting and important question is whether the treecost of a chordal graph equals its *f*-cost. We will see in Section 3 that this depends on the function *f*.

Definition. A function $f : \mathbf{N} \to \mathbf{R}^+$ is *fast*, if for all $i \in \mathbf{N}$, $f(i+1) \geqslant 2f(i)$.

An example of a fast function is the function $f(i) = 2^i$.

Definition. A tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of a graph $G = (V, E)$ is *minimal*, if there is no $\{i, j\} \in F$ with $X_i \subseteq X_j$.

It is well known that there is always a minimal tree decomposition of minimum treewidth. Such a minimal tree decomposition can be obtained by taking an arbitrary tree decomposition of minimum width, and while there is an edge $\{i, j\} \in F$ with $X_i \subseteq X_j$, contracting this edge, taking for the new node i', $X_{i'} = X_i \cup X_j = X_j$. The same construction can also be used for obtaining a minimal tree decomposition of minimum *f*-cost. Thus, we have the following lemma.

Lemma 3. *Let f be a function* $f : \mathbf{N} \to \mathbf{R}^+$.

(i) *Let* $(\{X_i \mid i \in I\}, T = (I, F))$ *be a tree decomposition of a graph* $G = (V, E)$ *of minimum f-cost. Then this tree decomposition is minimal*.
(ii) *Every graph G has a minimal tree decomposition with f-cost equal to the treecost of G with respect to f.*

Lemma 4. *Let f be a function* $f : \mathbf{N} \to \mathbf{R}^+$. *Let G be a graph with n vertices and with treewidth k. Then* $\mathrm{tc}_f(G) \leqslant (n-k)f(k+1)$.

Proof. Take a minimal tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of *G* of width *k*. This tree decomposition will have $|I| \leqslant n - k$, (see e.g. [3, Lemma 2.2]) and each node of the tree decomposition has at most $k + 1$ vertices. □

The following well-known lemma (see [6] for its proof) is used in some of our proofs.

Lemma 5. *Let* $(\{X_i \mid i \in I\}, T = (I, F))$ *be a tree decomposition of* $G = (V, E)$.

(i) *Suppose* $W \subseteq V$ *forms a clique in G. Then there is an* $i \in I$ *with* $W \subseteq X_i$.
(ii) *Suppose there are sets* $W_1, W_2 \subseteq V$, *such that for all* $w_1 \in W_1$, $w_2 \in W_2$, $\{w_1, w_2\} \in E$. *Then there is an* $i \in I$ *with* $W_1 \subseteq X_i$ *or* $W_2 \subseteq X_i$.

An alternative way of stating Lemma 5(ii) is:

Lemma 6. *Let H be a chordal supergraph of $G=(V,E)$, and suppose there are sets $W_1, W_2 \subseteq V$, such that for all $w_1 \in W_1$, $w_2 \in W_2$, $\{w_1, w_2\} \in E$. Then W_1 forms a clique in H or W_2 forms a clique in H.*

3. Minimal triangulations and treecost

In this section, we investigate for which chordal graphs and which functions *f*, the treecost equals the *f*-cost. Using the obtained results, we will see that for every fast function *f*, there always exists a minimal triangulation with optimal *f*-cost.

Lemma 7. *Let $f: \mathbf{N} \to \mathbf{R}^+$ be a function that is not fast. Then there is a chordal graph G, such that the f-cost of G is larger than the treecost of G with respect to f.*

Proof. Suppose $f(i+1) < 2f(i)$. Let G be the graph, obtained by taking a clique with $i+1$ vertices and remove one edge e. Then G has *f*-cost $2f(i)$, but the triangulation that is formed by adding the edge e has *f*-cost $f(i+1)$. □

Lemma 8. *Let $f: \mathbf{N} \to \mathbf{R}^+$ be a fast function. Let $G=(V,E)$ be a chordal graph. Let $v, w \in V$ with $\{v,w\} \notin E$. Let $G'=(V, E \cup \{v,w\})$ be a chordal graph. Then $\mathrm{cost}_f(G) \leqslant \mathrm{cost}_f(G')$.*

Proof. We associate to every maximal clique W of G' a set $h(W)$ of at most two maximal cliques of G. If $\{v,w\} \not\subseteq W$, then we take $h(W)=\{W\}$. Otherwise, if $\{v,w\} \subseteq W$, then we have that $W_1 = W - \{v\}$ and $W_2 = W - \{w\}$ are maximal cliques in G and we set $h(W) = \{W_1, W_2\}$.

Write $f(h(W)) = \sum_{Z \in h(W)} f(|Z|)$. We have that $f(h(W)) \leqslant \max\{f(|W|), 2 \cdot f(|W|-1)\} \leqslant f(|W|)$.

For every maximal clique W of G, there is a maximal clique W' of G' such that $W \in h(W')$. Note that $\{v,w\} \not\subseteq W$. If W is also a maximal clique of G', then $h(W)=\{W\}$. If W is not a maximal clique in G', then either $W \cup \{v\}$ or $W \cup \{w\}$ is a maximal clique in G', and $W \in h(W \cup \{v\})$ or $W \in h(W \cup \{w\})$.

Now, it follows that

$$\begin{aligned}
\mathrm{cost}_f(G') &= \sum_{W \subseteq V;\, W \text{ is a maximal clique in } G'} f(|W|) \\
&\geqslant \sum_{W \subseteq V;\, W \text{ is a maximal clique in } G'} f(h(W)) \\
&\geqslant \sum_{W \subseteq V;\, W \text{ is a maximal clique in } G} f(|W|) \\
&= \mathrm{cost}_f(G).
\end{aligned}$$

This proves the lemma. □

Lemma 9. *Let $G=(V,E_G)$ and $H=(V,E_H)$ be chordal graphs, and $f: \mathbf{N} \to \mathbf{R}^+$ be a fast function. Suppose G is a subgraph of H. Then $\mathrm{cost}_f(G) \leqslant \mathrm{cost}_f(H)$.*

Proof. We use induction on $|E_H - E_G|$. If $|E_H - E_G| = 0$, then $G=H$ and the result follows trivially.

Suppose the lemma holds for $|E_H - E_G| = i$. Now, suppose $|E_H - E_G| = i+1$. From [19, Lemma 2], it follows that there is an edge $e \in E_H - E_G$ such that $H' = (V, E_H - e)$ is chordal. By induction, $\mathrm{cost}_f(G) \leqslant \mathrm{cost}_f(H')$, and by Lemma 8, $\mathrm{cost}_f(H') \leqslant \mathrm{cost}_f(H)$, so $\mathrm{cost}_f(G) \leqslant \mathrm{cost}_f(H)$. □

Theorem 10. *Let $f: \mathbf{N} \to \mathbf{R}^+$ be a fast function. Every graph G has a minimal triangulation H, such that $\mathrm{cost}_f(H) = \mathrm{tc}_f(G)$.*

Proof. Suppose H' is a triangulation of G with $\mathrm{cost}_f(H') = \mathrm{tc}_f(G)$. H' contains a minimal triangulation H of G. Trivially, we have $\mathrm{cost}_f(H) \geqslant \mathrm{tc}_f(G)$. By the previous lemma, we have $\mathrm{cost}_f(H) \leqslant \mathrm{cost}_f(H')$. □

Corollary 11. *Let G be a chordal graph, and let f be a fast function. Then $\mathrm{cost}_f(G) = \mathrm{tc}_f(G)$.*

Proof. The only minimal triangulation of G is G itself. □

4. Separators

In this section, we obtain an important algorithmic consequence of Theorem 10. We show that for fast functions the treecost of graphs with a polynomial number of minimal separators can be computed efficiently. Our approach to this problem follows the ideas of Bouchitté and Todinca [8]. (See also Parra and Scheffler [18].) This allows one to find the treecost efficiently when the input is restricted to cocomparability graphs, d-trapezoid graphs, permutation graphs, circle graphs, weakly triangulated graphs and many other graph classes. See [9] for an encyclopedic survey on graph classes.

A subset S of vertices of a connected graph G is called an a, b-*separator* for non-adjacent vertices a and b in $V(G)\backslash S$ if a and b are in different connected component of the subgraph of G induced by $V(G)\backslash S$. If no proper subset of an a, b-separator S separates a and b in this way, then S is called a *minimal* a, b-*separator*. A subset S is referred to as a *minimal separator*, if there exist non-adjacent vertices a and b for which S is a minimal a, b-separator. Note that a minimal separator can be strictly contained in another minimal separator.

The following result of Dirac [11] is well known.

Theorem 12 (*Dirac [11]*). *Graph G is chordal if and only if every minimal separator of G is a clique*.

Lemma 13. *Let S be a minimal separator of a chordal graph G and* $\mathbf{C}$ *be the set of connected components in* $G\backslash S$. *Then for any fast function f*

$$\mathrm{tc}_f(G) = \sum_{C\in\mathbf{C}} \mathrm{tc}_f(G[N[C]]).$$

Proof. Since S is a minimal separator, we have that every vertex subset W is a maximal clique in G if and only if W is a maximal clique in *exactly* one of the graphs $G[N[C]]$. Therefore, $\mathrm{cost}_f(G) = \sum_{C\in\mathbf{C}} \mathrm{cost}_f(G[N[C]])$. By Theorem 10 this implies the proof of the lemma. □

Let Δ_G be the set of all minimal separators in G. Let $S \in \Delta_G$ be a minimal separator of a graph G. We denote by G_S the supergraph of G obtained from G by making all vertices of S adjacent. For a set of minimal separators $\Gamma \subseteq \Delta_G$ we denote by G_Γ the graph obtained from G by turning all separators from Γ into cliques.

There is a deep relation between the minimal separators of a graph and its minimal triangulations. We need the following generalization of Dirac's theorem by Parra and Scheffler [18].

Two separators S and T *cross* if there are distinct components C and D of $G\backslash T$ such that S intersects both of them. If S and T do not cross, they are called *parallel*.

Theorem 14 (*Parra and Scheffler [18]*). (i) *Let* $\Gamma \subseteq \Delta_G$ *be a maximal set of pairwise parallel separators of G. Then* $H = G_\Gamma$ *is a minimal triangulation of G and* $\Delta_H = \Gamma$. (ii) *Let H be a minimal triangulation of a graph G. Then* $\Gamma = \Delta_H$ *is a maximal set of pairwise parallel separators of G and* $H = G_\Gamma$.

Let S be a minimal separator of a graph G and $\mathbf{C}_S$ be the set of connected components of $G\backslash S$. A *block B* is a graph of the form $G_S[N[C]]$, where C is the vertex set of one of the connected components in $\mathbf{C}_G$. In other words, a block is obtained from a subgraph of G induced by vertex set C of a connected component of $G\backslash S$ and a subset $S_C = N[C] - C$ (vertices of S that are adjacent to at least one vertex in C) by adding a clique on S_C.

The following characterization of minimal separators is well-known (see e.g. [13, p. 106]).

Lemma 15. *Let S be an* a, b*-separator of G and let* G_a, G_b *be two components of* $G\backslash S$ *containing a and b, respectively. Then S is a minimal* a, b*-separator if and only if every vertex* $s \in S$ *is adjacent to a vertex in each of these components.*

Lemma 15 implies that for every minimal separator S, the set $\mathbf{B}_S$ contains at least two full blocks. Also Lemma 15 implies that for every block $B = (V_B, E_B) \in \mathbf{B}_S$ the set $V_B \cap S$ is a minimal separator and that B is a full block for $V_B \cap S$.

Theorem 16. *For any graph G and fast function f*

$$\mathrm{tc}_f(G) = \min_{S\in\Delta_G} \sum\nolimits_{B\in\mathbf{B}_S} \mathrm{tc}_f(B).$$

Proof. ($\leqslant$). Let S be a minimal separator and H_S be a minimal triangulation of G_S of optimal treecost. By Theorem 14 there is a minimal triangulation $H \subseteq H_S$ of G. By Lemma 9, $\mathrm{cost}_f(H) \leqslant \mathrm{cost}_f(H_S)$ and by Theorem 10, $\mathrm{tc}_f(G) \leqslant \mathrm{tc}_f(G_S)$.

For every block $B=(B_V,B_E)\in \mathbf{B}_S$, let $(\{X_i \mid i\in I_B\}, T_B=(I_B,F_B))$ be a tree decomposition of optimal treecost of this block. For every component C the vertices $N(C)\cap S$ induce a clique in B. Hence for every block $B\in \mathbf{B}_S$ and the corresponding tree $T_B=(I_B,F_B)$, there is a node $i_B\in I_B$ such that X_{i_B} contains all vertices of $B\cap S$. We choose one such node for every tree T_B. Moreover, by Lemma 15 there is a node i^* in some of the trees T_B such that the corresponding set X_{i^*} contains all vertices of S. We construct a tree decomposition of G_S with treecost $\sum_{B\in\mathbf{B}_S}\mathrm{tc}_f(B)$ from the tree decompositions of blocks $\mathbf{B}_S$. The tree of this decomposition is obtained by taking disjoint union of trees T_B and making node i^* adjacent to nodes i_B, $B\in\mathbf{B}_S$. One can check easily that this is a tree decomposition of G_S. The cost of this decomposition is equal to the sum of the costs of B. Therefore, $\mathrm{tc}_f(G)\leqslant \mathrm{tc}_f(G_S)\leqslant\sum_{B\in\mathbf{B}_S}\mathrm{tc}_f(B)$.

($\geqslant$). Let H be a minimal triangulation of G such that $\mathrm{tc}_f(H)=\mathrm{tc}_f(G)$. Let S be a minimal separator of H. By Lemma 13, we have that $\mathrm{tc}_f(H)=\sum_{C\in\mathbf{C}_S}\mathrm{tc}_f(H[N[C]])$. For every $C\in\mathbf{C}_S$ the corresponding block $B\in\mathbf{B}_S$ is the induced subgraph of $H[N[C]]$ and hence chordal. Then by Theorem 10

$$\mathrm{tc}_f(G)=\mathrm{tc}_f(H)=\sum_{C\in\mathbf{C}_S}\mathrm{tc}_f(H[N[C]])\geqslant\sum_{B\in\mathbf{B}_S}\mathrm{tc}_f(B).$$

By Theorem 14, S is also minimal separator of G. Therefore, $\sum_{B\in\mathbf{B}_S}\mathrm{tc}_f(B)\geqslant\min_{S\in\Delta_G}\sum_{B\in\mathbf{B}_S}\mathrm{tc}_f(B)$. $\square$

Vertex set $\Omega\subseteq V$ of a graph G is a *potential maximal clique* if there is a minimal triangulation H of G such that Ω is a maximal clique in H. We denote by Π_G the set of all potential maximal cliques in G. Bouchitté and Todinca [7] proved that $|\Pi_G|=\mathrm{O}(n|\Delta_G|^2)$ and that the potential maximal cliques can be computed in polynomial time in size of the graph and the number of its minimal separators.

Let Ω be a potential maximal clique of G. Let $C_1, C_2, \ldots, C_k$ be the connected components of $G\backslash\Omega$. By Lemma 15, $\Omega\cap C_i$ are minimal separators and the graphs $G_\Omega[N[C_i]]$ are blocks. We call these blocks the *blocks associated with* Ω. The set of all blocks associated with potential clique Ω is denoted by $\mathbf{B}_\Omega$.

The following result was obtained by Bouchitté and Todinca.

Theorem 17 (*Bouchitté and Todinca [8]*). *Let $B=(V_B,E_B)$ be one of the full blocks of G corresponding to minimal separator S. Then $H=(V_H,E_H)$ is a minimal triangulation of B if and only if there is a potential maximal clique $\Omega\subseteq V_B$ (maximal clique of G) such that*

- *$S\subset\Omega$;*
- *H is obtained from B by turning Ω into a clique and taking minimal triangulations of blocks in B associated with Ω. More precisely, let $B_1=(V_1,E_1),\ldots,B_k=(V_k,E_k)$ be the blocks from $\mathbf{B}_\Omega$ in B associated with Ω. Then $V_H=V_1\cup\cdots\cup V_k\cup\Omega$ and $E_H=E_1\cup\ldots\cup E_k\cup\{\{x,y\}: x,y\in\Omega\}$.*

As a consequence, we have the following result.

Theorem 18. *Let $B=(V_B,E_B)$ be a full block of G corresponding to a minimal separator S, let f be a fast function. Then*

$$\mathrm{tc}_f(B)=\min_{S\subset\Omega\subseteq V_B,\Omega\in\Pi_G}\left(f(|\Omega|)+\sum_{B_i\in\mathbf{B}_\Omega}\mathrm{tc}_f(B_i)\right).$$

Proof. Let H be a minimal triangulation of B with optimal treecost. Then by Theorem 17 there is a potential maximal clique $S\subset\Omega$ such that H is obtained by turning Ω into clique and taking minimal triangulations $H_1, H_2, \ldots, H_k$ of blocks in B associated with Ω.

By the definition of blocks associated with clique Ω, every clique $W\neq\Omega$ in H is maximal if and only if W is a maximal clique in exactly one triangulation H_i. Then

$$\mathrm{tc}_f(B)=\mathrm{cost}_f(H)=f(|\Omega|)+\sum_{B_i\in\mathbf{B}_\Omega}\mathrm{cost}_f(H_i)\geqslant f(|\Omega|)+\sum_{X_i\in\mathbf{B}_\Omega}\mathrm{tc}_f(B_i).$$

In the other direction, let Ω be a maximal potential clique and let H_i be minimal triangulations of $B_i\in\mathbf{B}_\Omega$ with minimum f-cost. Let H be the triangulation of B obtained by turning Ω into clique and taking triangulations $H_1, H_2, \ldots, H_k$ as triangulations of the corresponding associated blocks. The f-cost of H is at most $f(|\Omega|)+\sum_{B_i\in\mathbf{B}_\Omega}\mathrm{cost}_f(H_i)$. By Theorem 17, H is a minimal triangulation and by Theorem 10, $\mathrm{tc}_f(B)\leqslant\mathrm{cost}_f(H)$. $\square$

Theorem 19. *Let f be a fast function and let $T_f(n)$ be the time needed to compute $f(1), \ldots, f(n)$. Let Δ_G be the set of all minimal separators in G. Then for every graph G there exists an $\mathrm{O}(n^2|\Delta_G|^3 + T_f(n) + n^2 m|\Delta_G|^2)$ time algorithm for computing the treecost of G.*

Proof. To prove the theorem we present the algorithm similar to the algorithm for treewidth and fill-in by Bouchitté and Todinca [8].

INPUT: G and all its minimal separators.
OUTPUT: $\mathrm{tc}_f(G)$

(1) Use Bouchitté–Todinca's algorithm [7] to compute all potential maximal cliques of G;
(2) For every minimal separator compute the set of blocks $\mathbf{B}_S$ and sort all blocks by the number of vertices;
(3) For every block $B = (V_B, E_B)$ (and the corresponding minimal separator S) in order of increasing size do
  - $\mathrm{tc}_f(B) := \infty$;
  - For every potential maximal clique Ω such that $S \subset \Omega \subseteq V_B$; compute the blocks $\mathbf{B}_\Omega$ associated with Ω;
  - $\mathrm{tc}_f(B) := \min(\mathrm{tc}_f(B), f(|\Omega|) + \sum_{X \in \mathbf{B}_\Omega} \mathrm{tc}_f(X))$;

(4) $\mathrm{tc}_f(G) = \min_{S \in \Delta_G} \sum_{B \in \mathbf{B}_S} \mathrm{tc}_f(B)$.

The correctness of the algorithm follows from Theorems 10 and 18.

The running time of the first step of the algorithm is $\mathrm{O}(n^2 m|\Delta_G|^2)$ (see [7]). Let b be the number of blocks in G. Because for every minimal separator S the set $\mathbf{B}_S$ has cardinality at most n, we have that $b \leqslant n|\Delta_G|$ and the second step can be implemented in $\mathrm{O}(n|\Delta_G| + mn)$ time. The third step can be implemented in $\mathrm{O}(b|\Pi_G| + T_f(n)) = \mathrm{O}(n|\Delta_G||\Pi_G| + T_f(n)) = \mathrm{O}(n^2|\Delta_G|^3 + T_f(n))$ time. □

5. Cographs

In this section, we give a relatively simple algorithm that computes the treecost of a cograph with respect to a function *f*, and constructs the corresponding tree decomposition. When $f(1), \ldots, f(n)$ can be computed in linear time, the algorithm uses linear time. A polynomial time algorithm for the problem can be obtained from Theorem 19, as cographs are a subclass of the permutation graphs and have polynomially many minimal separators; the algorithm given in this section is faster and simpler, and also works for functions *f* that are not fast.

The algorithm follows the same pattern as many algorithms on cographs, and uses ideas of the algorithm to compute the treewidth of a cograph from [6]. Let $f \circ +j$ denote the function with for all $i \in \mathbf{N}$: $(f \circ +j)(i) = f(i + j)$. Any cograph can be formed from graphs with one vertex by the following operations: disjoint union and product ($\times$), where the product of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is formed by taking the disjoint union of G_1 and G_2 and then adding all $|V_1| \cdot |V_2|$ edges between the vertices in V_1 and the vertices in V_2.

Lemma 20. *Let $f : \mathbf{N} \to \mathbf{R}^+$ be a function. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be disjoint graphs.*
(i) $\mathrm{tc}_f(G_1 \cup G_2) = \mathrm{tc}_f(G_1) + \mathrm{tc}_f(G_2)$.
(ii) $\mathrm{tc}_f(G_1 \times G_2) = \min\{\mathrm{tc}_{f \circ +|V_2|}(G_1), \mathrm{tc}_{f \circ +|V_1|}(G_2)\}$.

Proof. (i) Trivial.

(ii) If we take a triangulation H_1 of G_1 with minimum $(f \circ +|V_2|)$-cost, and then turn V_2 into a clique, we obtain a triangulation H of $G_1 \times G_2$. For every maximal clique W in H, $W - V_2$ is a maximal clique in H_1, and hence the *f*-cost of H is $\mathrm{tc}_{f \circ +|V_2|}(G_1)$. Similarly, we can make a triangulation of $G_1 \times G_2$ of *f*-cost $\mathrm{tc}_{f \circ +|V_1|}(G_2)$.

Suppose H is a triangulation of $G_1 \times G_2$ such that $\mathrm{cost}_f(H)$ is minimal. Then by Lemma 6, either V_1 or V_2 forms a clique in H. Suppose V_1 is a clique in H. Let H_2 be the triangulation of G_2 obtained by restricting H to G_2. As for every maximal clique W in H_2, we have that $W \cup V_1$ is a maximal clique in H, we have that $\mathrm{tc}_f(H) = \mathrm{cost}_f(H) = \mathrm{cost}_{f \circ +|V_1|}(H_2)$. So in this case, $\mathrm{tc}_f(H) \geqslant \mathrm{tc}_{f \circ +|V_1|}(G_2)$. If V_2 forms a clique in H, then similarly, $\mathrm{tc}_f(H) \geqslant \mathrm{tc}_{f \circ +|V_2|}(G_1)$. □

As one can find in $\mathrm{O}(|V| + |E|)$ time, a series of disjoint union and product operations that build a given cograph [10], the following result can be obtained similar to many other algorithmic results on cographs:

Theorem 21. *Let $f : \mathbf{N} \to \mathbf{R}^+$ be a function. Let $T_f(n)$ be the time needed to compute $f(1), \ldots, f(n)$. Then there is an algorithm that computes $\mathrm{tc}_f(G)$ for a given cograph with n vertices and m edges in $\mathrm{O}(n + m + T_f(n))$ time.*

Note that we do not need that *f* is fast.

6. Graphs of treewidth two

For graphs of treewidth at most two it holds that there always exists a triangulation of minimum *f*-cost that also has minimum treewidth (i.e., treewidth two), assuming that *f* is fast.

Lemma 22. *For any fast function f and any graph G, the treecost of G with respect to f equals the sum over the biconnected components of G of the treecost of the components with respect to f.*

Proof. If we have a triangulation of each biconnected component of *G*, then taking these together gives a triangulation of *G*; noting that each maximal clique of that triangulation appears once as a maximal clique in a triangulation of a biconnected component shows that the treecost of *G* is at most the sum over the biconnected components of their treecosts.

Suppose we have a triangulation *H* of *G* of minimum *f*-cost. By Theorem 10, we may assume that *H* is a minimal triangulation. Hence, *H* does not contain edges between different biconnected components of *G*; the biconnected components of *H* have the same vertex sets as the biconnected components of *G*. Thus, the sum of the *f*-costs of the triangulations, obtained by restricting *H* to the different biconnected components equals the *f*-cost of *H*. □

Lemma 23. *Let G be a biconnected chordal graph. Then every maximal clique in G has size at least three.*

Proof. Suppose $\{v, w\}$ is a maximal clique of size two in biconnected chordal graph *G*. If *v* and *w* have no common neighbor in *G*, then we easily can construct a chordless cycle of length more than three in *G*. □

Lemma 24. *Let $G = (V, E)$ be a biconnected graph of treewidth at most two. Let f be a fast function. Let $n = |V|$. If $n = 2$, $\mathrm{tc}_f(G) = f(2)$, and if $n \geqslant 3$, $\mathrm{tc}_f(G) = f(3) \cdot (n-2)$.*

Proof. If $n = 2$, then *G* consists of a single edge, and clearly $\mathrm{tc}_f(G) = f(2)$.

We use induction on *n* for the case $n \geqslant 3$. If $n = 3$, then *G* is isomorphic to K_3: a clique with three vertices, and hence $\mathrm{tc}_f(G) = f(3)$. Suppose the lemma is true upto $n - 1$. Let $G = (V, E)$ be a biconnected graph with $n \geqslant 4$ vertices and treewidth at most two. By Lemma 4, we have that $\mathrm{tc}_f(G) \leqslant f(3) \cdot (n-2)$.

Suppose we have a triangulation *H* of *G* of optimal *f*-cost. Consider a vertex *v* that is simplicial in *H*. If $N_H(v)$ is not a maximal clique in $H - v$, then $\mathrm{tc}_f(G) = f(|N_H(v)| + 1) + \mathrm{tc}_f(H - v) \geqslant f(3) + (n-2) f(3)$. If $N_H(v)$ is a maximal clique in $H - v$, then, by Lemma 23, $|N_H(v)| \geqslant 3$, and hence $\mathrm{tc}_f(G) = f(|N_H(v) \cup \{v\}|) + \mathrm{tc}_f(H - v) - f(|N_H(v)|) \geqslant f(|N_H(v)|) + (n - 2) f(3) \geqslant (n-1) f(3)$. (We have used in this step that *f* is fast.) □

The proof of the preceding lemma shows that any triangulation of a biconnected graph of treewidth two with maximum clique size three has optimal *f*-cost; *f* any fast function. Such a triangulation can be easily obtained by taking a vertex *v* of degree two, making its neighbors adjacent, recursively triangulating the graph without *v*, and then adding *v* back. This is similar to the algorithm to recognize graphs of treewidth two, see [1]. For an arbitrary (not necessarily biconnected) graph *G* of treewidth at most two, we can apply this procedure for every biconnected component separately.

Theorem 25. *Let f be a fast function, such that $f(1)$, $f(2)$, and $f(3)$ are computable. Then there is a linear time algorithm that computes the treecost with respect to f of a graph of treewidth at most two.*

7. Treewidth versus treecost

An interesting question is whether there is always a triangulation with both optimal treecost and with optimal treewidth. Such a result would have had nice practical algorithmic consequences (e.g., in the algorithm of Section 4, we can ignore all separators larger than the treewidth plus one). Unfortunately, such triangulations do not always exist. In the example, given in Fig. 1, we have a cograph that is formed as follows. G_1 is the disjoint union of four triangles (copies of K_3). G_2 is the disjoint union of a clique with four vertices and eight isolated vertices. *G* is the product of G_1 and G_2. Let *f* be the function $f(n) = 2^n$. Now, a triangulation of minimum treewidth is obtained by turning V_2 into a clique: this gives a maximum clique size of 15 (whereas when we turn V_1 into a clique, we have a triangulation with maximum clique size 16.) A triangulation of $G_1 \times G_2$ of minimum *f*-cost is obtained by turning V_1 into a clique: this gives an *f*-cost of $2^{12}(2^4 + 8)$; turning V_2 into a clique gives an *f*-cost of $2^{12}(4 \times 2^3)$.

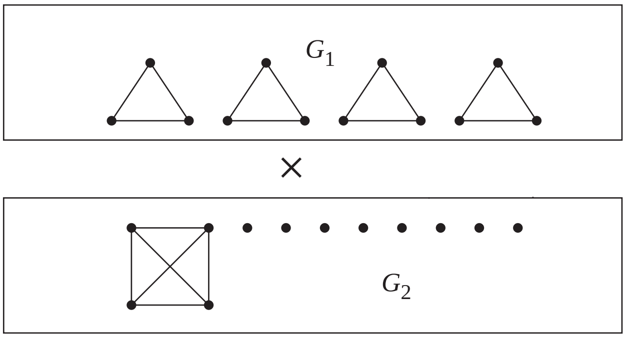

Fig. 1. A cograph whose triangulation with optimal treecost has not optimal treewidth.

More generally, let $\mathrm{tc}_{f,k}(G)$ be the minimum *f*-cost of a tree decomposition of *G* of width at most *k*. The cograph given above is an example of a graph where $\mathrm{tc}_{f,k}(G) \neq \mathrm{tc}_f(G)$, *k* the treewidth of *G*.

We conjecture that the width of a tree decomposition of optimal *f*-cost cannot be 'much' larger than the treewidth of a graph:

Conjecture 26. *Let f be a fast function. There exists a function g_f, such that for all graphs G of treewidth at most k,* $\mathrm{tc}_f(G) = \mathrm{tc}_{f,g_f(k)}(G)$.

Having such a function g_f would help to speed up the algorithm of Section 4. A proof of Conjecture 26 would imply that for every polynomial time computable fast function, the treecost of graphs of bounded treewidth is polynomial time computable, because we have the following result.

Theorem 27. *Let $f : \mathbf{N} \to \mathbf{R}^+$ be function, such that for each n, $f(n)$ can be computed. Let $k \in \mathbf{R}^+$. There exists an algorithm that computes for a given graph G, $\mathrm{tc}_{f,k}(G)$ in $\mathrm{O}(n^{k+2})$ time, plus the time needed to compute $f(1), \ldots, f(k+1)$.*

Proof. We sketch the proof here. Let Π_G^{k+1} be the set of all potential maximal cliques in *G* of cardinality at most $k+1$. Similar to the proof of Theorem 18 one can prove the following: Let $B = (V_B, E_B)$ be a full block of *G* corresponding to minimal separator *S*. Then

$$\mathrm{tc}_{f,k}(G) = \min_{S \subset \Omega \subseteq V_B, \Omega \in \Pi_G^{k+1}} \left(f(|\Omega|) + \sum_{B_i \in \mathbf{B}_\Omega} \mathrm{tc}_{f,k}(B_i) \right).$$

The results of Bouchitté and Todinca [7] imply that for a vertex set *K* one can recognize in $\mathrm{O}(|K|m)$ time if *K* is a potential maximal clique. If $m > kn$, then *G* has treewidth more than *k* (see [4]), and hence $\mathrm{tc}_{f,k}(G) = \infty$. So, we may assume that we have a linear number of edges. Therefore, in our case, a potential maximal clique of size at most $k+1$ can be recognized in $\mathrm{O}(n)$ time and the set Π_G^{k+1} can be computed in $\mathrm{O}(n^{k+2})$ time.

Checking if a given set is a separator can be done in $\mathrm{O}(n)$ time, so finding the list of minimal separators of size at most *k* costs $\mathrm{O}(n^{k+1})$ time. (By Theorems 14 and 16 only minimal separators of size at most *k* have to be considered.)

Now one can use the modified version of the algorithm in the proof of Theorem 19 restricted to the set of potential maximal cliques of sizes at most $k+1$ and minimal separators of size at most *k* to obtain $\mathrm{tc}_{f,k}(G)$. □

There is also a constructive variant of the algorithm (it outputs the desired tree decomposition) that runs also in $\mathrm{O}(n^{k+2})$ time.

8. Probabilistic networks and vertex weights

Probabilistic networks are the underlying technology of several modern decision support systems. See e.g. [14]. Such a probabilistic network models independencies and dependencies between statistical variables with help of a directed acyclic graph. A central problem is the PROBABILISTIC INFERENCE problem: one must determine the probability distribution of a specific variable, possibly given the values of some other variables. As this problem is $\#P$-complete for general networks [20] but many networks used in practice appear to have small treewidth, an algorithm of Lauritzen and Spiegelhalter [17] is often used that solves the problem on networks with small treewidth.[1] As the same network is used for many computations, it is very useful to spend much preprocessing time and obtain a tree decomposition that allows fast computations. Thus, more important than minimizing the width is to minimize the 'cost' of the tree decomposition. While each vertex models a discrete statistical variable, variables may have a different valence. Let $w(v) \in \mathbf{N}$ be the *weight* of v. $w(v)$ models the number of values v can take, which directly reflects on the resources (time and space) needed for a computation. For instance, a binary variable corresponds to a vertex with weight two. In a tree decomposition of G, the time to process a node is basically the product of the weights of the vertices in the corresponding set X_i. In graph terms, we can model the situation as follows, after [16,22,15].

Given are a graph $G = (V, E)$, and a weight function $w : V \to \mathbf{N}$. The *total state space* of a triangulation H of G is the sum over all maximal cliques W in H of $\prod_{v \in W} w(v)$.

Note that when all vertices have weight two (i.e., all variables are binary), then the total state space is exactly the f-cost with for all i, $f(i) = 2^i$.

Some of the proofs of previous sections can be modified to give similar results for the problem to find a triangulation of minimum total state space.

Theorem 28.

(i) *Let G be a graph, with vertices weighted with positive integers. Then there is a minimal triangulation H with total state space equal to the minimum total state space of a triangulation of G.*
(ii) *There exists an algorithm to compute a triangulation with minimum total state space whose running time is polynomial in the number of minimal separators of G.*
(iii) *Given a cograph G with vertices weighted with positive integers, a triangulation of G with minimum total state space can be found in linear time.*
(iv) *For each k, there is an algorithm that runs in* $O(n^{k+2})$ *time, and that given a graph G with vertices weighted with positive integers, finds among the tree decompositions of G of width at most k finds one of minimum state space.*

The method to compute the treecost of a graph of treewidth two of Section 6 cannot be used for the minimum state space problem when vertices have different weights.

9. Hardness results

Wen [22] showed that TREECOST$_f$ is NP-hard when f is the function $f(i) = 2^i$. To be precise, Wen showed that the problem of finding a triangulation of minimum total state space is NP-hard when all variables are binary. In this section, we show similar results for a larger class of functions f, using a different reduction, and we show that the problems remain NP-hard for cobipartite and for bipartite graphs.

Theorem 29. *Let f be a fast function. The* TREECOST$_f$ *problem is NP-hard for cobipartite graphs.*

Proof. We reduce from TREEWIDTH. Let an instance of the TREEWIDTH problem be given: a graph $G = (V, E)$ and an integer $k \leqslant |V|$.

We transform G to a graph H as follows: for every $v \in V$, we take $\log n$ vertices $v_1, \ldots, v_{\log n}$; and for every edge $\{v, w\} \in E$, we take the edges $\{v_i, w_j\}$ for all i, j, $1 \leqslant i \leqslant \log n$, $1 \leqslant j \leqslant \log n$. In addition, we add edges $\{v_i, v_j\}$ for all $1 \leqslant i < j \leqslant \log n$. $\square$

Claim 1. *The treewidth of G is at most k, if and only if the treecost of H is at most* $(n - 1)\, f((k + 1) \log n)$.

Proof. Suppose we have a minimal tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of G of width at most k.

Taking $Y_i = \{v_j \mid v \in X_i, 1 \leqslant j \leqslant \log n\}$, we have that $(\{Y_i \mid i \in I\}, T = (I, F))$ is a tree decomposition of f-cost at most $(n - 1) \cdot f((k + 1) \log n)$.

Now, suppose $(\{Y_i \mid i \in I\}, T = (I, F))$ is a tree decomposition of minimum f-cost of H. By Lemma 3, we assume that this tree decomposition is minimal. Take for all $i \in I$: $X_i = \{v \in V \mid v_1, \ldots, v_{\log n} \in Y_i\}$. One can verify that $(\{X_i \mid i \in I\}, T = (I, F))$ is a tree decomposition of G. (The second condition of tree decomposition can be seen to hold as follows: for every edge $\{v, w\} \in E$, the set $\{v_1, \ldots, v_{\log n}, w_1, \ldots, w_{\log n}\}$ forms a clique in H, hence there is an $i \in I$ with $\{v_1, \ldots, v_{\log n}, w_1, \ldots, w_{\log n}\} \subseteq Y_i$

[1] To be precise, first the moralization of the network is made: for every vertex, the set of its direct predecessors is turned into a clique, and then all directions of edges are dropped.

(Lemma 5), hence $v, w \in X_i$.) The width of this decomposition is at most *k*: if there is an $i \in I$ with $|X_i| \geqslant k+2$, then $|Y_i| \geqslant (k+2) \log n$, and hence the *f*-cost of the tree decomposition of *H* is at least $f((k+2)\log n) \geqslant 2^{\log n} \cdot f((k+1)\log n) > (n-1) \cdot f((k+1)\log n)$. Hence, we have a tree decomposition of *G* of width at most *k*. □

Note that if *G* is a cobipartite graph, then *H* is a cobipartite graph. As we can construct *H* from *G* in polynomial time, the NP-completeness result now follows.

Theorem 30. *Let f be a fast function. The* TREECOST$_f$ *problem is NP-hard for bipartite graphs.*

Proof. Let *G* and *H* be as in the previous proof, but instead replace every vertex in *G* by $2 \log n$ vertices; and let H' be obtained from *H* by subdividing every edge. □

Claim 2. *The treewidth of G is at most k, if and only if the treecost of H' is at most* $(n-1) f((k+1)2 \log n) + 4 f(3) n^2 \log^2 n$.

Proof. Make a tree decomposition of *H* as in the proof of the previous theorem.

Suppose the treewidth of *G* is at most *k*. For each of the at most $4n^2 \log^2 n$ subdivision vertices in H', we have that *H* contains an edge between its neighbors, and hence we can add a set X_v, containing *v* and its neighbors and make it adjacent to a set that contains the neighbors of *v*. This gives a tree decomposition of H' of *f*-cost at most $(n-1) f((k+1)2 \log n) + 4 f(3) n^2 \log^2 n$.

Suppose the treecost of H' is at most $(n-1) f((k+1)2 \log n) + 4 f(3) n^2 \log^2 n$. Build a tree decomposition of *G* as in the proof for cobipartite graphs. Note that $f((k+2)2 \log n) \geqslant 2^{2 \log n} f((k+1)2 \log n) > (n-1) f((k+1) \log n) 4 f(3) n^2 \log^2 n$, so we must have that this tree decomposition has width at most *k*. □

Finally, note that *H* is bipartite when *G* is bipartite, and that *H* can be constructed in polynomial time from *G*. The theorem now follows from the fact that TREEWIDTH is NP-complete for bipartite graphs. □

Corollary 31. *Let f be a fast function such that there is an algorithm that computes for each n, $f(n)$ in time polynomial in n. Then the* TREECOST$_f$ *problem is NP-complete for cobipartite graphs and for bipartite graphs.*

In [5], it was shown that there is no algorithm that approximates the treewidth within a constant additive term unless $P = \mathrm{NP}$. Combining this result with the proof technique of the NP-hardness results given above can be used to show:

Theorem 32. *If $P \neq \mathrm{NP}$, then for every $c \in \mathbf{N}$, there is no polynomial time algorithm that approximates the treecost of a given graph G within a multiplicative factor c.*

10. Discussion

In this paper, we investigated a notion that gives a more refined view on what is a 'good' tree decomposition of a graph. For several algorithms on tree decompositions, the function that maps a tree decomposition to the amount of time spent by the algorithm when using that tree decomposition is actually somewhat more complicated than the *f*-costs as used in this paper, but the *f*-cost functions come close to these exact models. For instance, one can observe that often the degree of nodes in *T*, and the differences in set sizes along edges in *T* also influence the running time.

In some cases, the *f*-costs equals the space used by the algorithm, apart from a negligible small amount of space for the actual representation of the tree decomposition and control variables used by the algorithm. When running a dynamic programming algorithm one can often reuse memory. In [2], it was investigated how tree decompositions can be obtained that require little memory, when reusing memory. (See also [21].) However, that approach is mainly useful when solving decision problems. When solving the construction variant of the problem, one generally wants to keep all tables (as reusing space of tables means that one might have to recompute some when constructing solutions for yes-instances of the problem). In the latter case, the *f*-cost expresses precisely the space used by all the tables computed by the dynamic programming algorithm. We think it is an interesting and important problem to study notions that more precisely reflect the time used by algorithms using tree decompositions, and investigate their algorithmic complexity.

In other cases, the *f*-cost of the tree decomposition can represent the amount of space needed for the algorithm, in particular, the total size of all tables a specific dynamic programming algorithm uses with the tree decomposition.

We have seen that in several interesting cases, tree decompositions with optimal *f*-cost can be computed in polynomial time, and we expect that in some practical cases, where it makes sense to spend sufficiently much preprocessing time on finding one good tree decomposition (in particular, in cases, where the same tree decomposition is used several times with different data on the same graph or network), some of our methods can be of practical use.

Acknowledgements

We thank Linda van der Gaag for discussions about probabilistic networks and introducing the notion of total state space to us. We thank an anonymous referee for a shorter and more elegant proof of Lemma 9.

References

[1] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in a *k*-tree, SIAM J. Algebraic Discrete Methods 8 (1987) 277–284.
[2] B. Aspvall, A. Proskurowski, J.A. Telle, Memory requirements for table computations in partial *k*-tree algorithms, Algorithmica 27 (2000) 382–394.
[3] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (1996) 1305–1317.
[4] H.L. Bodlaender, A partial *k*-arboretum of graphs with bounded treewidth, Theoret. Comput. Sci. 209 (1998) 1–45.
[5] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, T. Kloks, Approximating treewidth, pathwidth, frontsize, and minimum elimination tree height, J. Algorithms 18 (1995) 238–255.
[6] H.L. Bodlaender, R.H. Möhring, The pathwidth and treewidth of cographs, SIAM J. Discrete Math. 6 (1993) 181–188.
[7] V. Bouchitté, I. Todinca, Listing all potential maximal cliques of a graph, in: H. Reidel, S. Tison (Eds.), Proceedings STACS'00, Lecture Notes in Computer Science, Springer, Berlin, vol. 1770, 2000, pp. 503–515.
[8] V. Bouchitté, I. Todinca, Treewidth and minimum fill-in: grouping the minimal separators, SIAM J. Comput. 31 (1) (2001).
[9] A. Brandstädt, V.B. Le, J.P. Spinrad, Graph Classes: A Survey, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.
[10] D.G. Corneil, Y. Perl, L.K. Stewart, A linear recognition algorithm for cographs, SIAM J. Comput. 14 (1985) 926–934.
[11] G. Dirac, On rigid circuit graphs, Abh. Math. Sem. Univ. Hamburg 25 (1961) 71–76.
[12] F. Gavril, The intersection graphs of subtrees in trees are exactly the chordal graphs, J. Combin. Theory Ser. B 16 (1974) 47–56.
[13] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.
[14] F.V. Jensen, Bayesian Networks and Decision Graphs, Statistics for Engineering and Information Science, Springer, New York, 2001.
[15] U. Kjærulff, Triangulation of graphs—algorithms giving small total state space, Research Report R-90-09, Department of Mathematics and Computer Science, Aalborg University, 1990.
[16] U. Kjærulff, Optimal decomposition of probabilistic networks by simulated annealing, Statist. Comput. 2 (1992) 2–17.
[17] S.J. Lauritzen, D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, J. Roy. Statist. Soc. Ser. B (Methodological) 50 (1988) 157–224.
[18] A. Parra, P. Scheffler, How to use the minimal separators of a graph for its chordal triangulation, in: Automata, Languages and Programming (Szeged, 1995), Springer, Berlin, 1995, pp. 123–124.
[19] D.J. Rose, R.E. Tarjan, G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput. 5 (1976) 266–283.
[20] D. Roth, On the hardness of approximate reasoning, Artificial Intelligence 82 (1996) 273–302.
[21] J.A. Telle, Tree-decompositions of small pathwidth, Technical report, Department of Informatics, University of Bergen, Bergen, Norway, 2001. Discr. Appl. Math., doi:10.1016/j.dam.2004.01.012.
[22] W.X. Wen, Optimal decomposition of belief networks, in: P.P. Bonissone, M. Henrion, L.N. Kanal, J.F. Lemmer (Eds.), Proceedings of the Sixth Workshop on Uncertainty in Artificial Intelligence, 1990, pp. 245–256.