A Generalization of AT-Free Graphs and a Generic Algorithm for Solving Treewidth, Minimum Fill-In and Vertex Ranking

Hajo Broersma[1], Ton Kloks[1], Dieter Kratsch[2], and Haiko Müller[2]

[1] Faculty of Applied Mathematics
University of Twente, P.O. Box 217
7500 AE Enschede
the Netherlands
{H.J.Broersma,A.J.J.Kloks}@math.utwente.nl
[2] Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität
07740 Jena
Germany
{kratsch,hm}@minet.uni-jena.de

Abstract. A subset A of the vertices of a graph G is an *asteroidal set* if for each vertex $a \in A$, the set $A \setminus \{a\}$ is contained in one component of $G - N[a]$. An asteroidal set of cardinality three is called *asteroidal triple* and graphs without an asteroidal triple are called *AT-free*. The maximum cardinality of an asteroidal set of G, denoted by $\mathsf{an}(G)$, is said to be the *asteroidal number* of G. We present a scheme for designing algorithms for triangulation problems on graphs. As a consequence, we obtain algorithms to compute graph parameters such as treewidth, minimum fill-in and vertex ranking number. The running time of these algorithms is a polynomial (of degree asteroidal number plus a small constant) in the number of vertices and the number of minimal separators of the input graph.

1 Introduction

Graphs without an asteroidal triple are called asteroidal triple-free graphs (short AT-free graphs) and attained much attention recently. Möhring has shown that every minimal triangulation of an AT-free graph is an interval graph which implies that for every AT-free graph the treewidth and the pathwidth of the graph are equal [25]. Furthermore a collection of interesting structural and algorithmic properties of AT-free graphs has been obtained by Corneil, Olariu and Stewart, among them an existence theorem for so-called dominating pairs in connected AT-free graphs and a linear time algorithm to compute a dominating pair for connected AT-free graphs (see [10,11]).

The class of graphs with bounded asteroidal number extends the class of AT-free graphs, based on a natural way of generalizing the concept of asteroidal triples to so-called asteroidal sets, first given by Walter [29]. A set of vertices

A of a graph G is called an asteroidal set if for every vertex $a \in A$ all vertices of $A \setminus \{a\}$ are contained in the same component of $G - N[a]$. Walter, Prisner and Lin et al. used asteroidal sets to characterize certain subclasses of the class of chordal graphs [23,27,29]. We introduce the asteroidal number of a graph G, denoted by $\mathsf{an}(G)$, as the maximum cardinality of an asteroidal set in G. Thus AT-free graphs are exactly those graphs G with $\mathsf{an}(G) \leq 2$.

In this paper we consider the NP-complete graph problems TREEWIDTH, MINIMUM FILL-IN and VERTEX RANKING that all remain NP-complete when restricted to AT-free graphs. In fact, each of the three problems remains NP-complete on cobipartite graphs [2,6,30], that form a small subclass of the class of AT-free graphs. TREEWIDTH has been studied in numerous recent papers, mainly since many NP-complete graph problems become solvable in polynomial time or even linear time when restricted to the class of graphs with bounded treewidth [1,4,16]. In this respect it is interesting, that for each constant k, there is a linear time algorithm that determines whether a given graph has treewidth at most k [5,16]. The MINIMUM FILL-IN problem stems from the optimal performance of Gaussian elimination on sparse matrices and has important applications in this area. Both TREEWIDTH and MINIMUM FILL-IN ask for a certain chordal embedding of the given graph. This often allows the design of similar algorithms for both problems, when graphs of some special class are considered. The VERTEX RANKING problem received much attention lately because of the growing number of applications. The problem of finding an optimal vertex ranking is equivalent to the problem of finding a minimum-height elimination tree of a graph [12]. This measure is of importance for the parallel Cholesky factorization of matrices [7,24]. Other applications lie in the field of VLSI-layout design [21].

Using an algorithm of [17] to list all minimal separators of a given graph G in time $O(n^5 r)$, where n is the number of vertices of G and r is the number of minimal separators of G, one has designed algorithms with a running time bounded by a polynomial in the number of vertices and the number of minimal separators of the input graph, that compute the treewidth and the minimum fill-in [19] as well as the vertex ranking number [20] on AT-free graphs. It is worth mentioning that the running time of these algorithms is not bounded by a polynomial in the input length, since AT-free graphs may have 'exponentially' many minimal separators.

We generalize the method used in [19,20]. To be more precise, we focus on certain sets of minimal separators called blocking sets. We show that these blocking sets have at most $\mathsf{an}(G)$ elements, and that they decompose the graph into a number of so-called blocks, which is bounded by a polynomial of order $\mathsf{an}(G)$ in the number of minimal separators of G. We consider graphs H obtained from a block of G by making the separators of the blocking set complete, and establish a relation between the blocks of H and the blocks of G. Together with some known recurrence relations for the three aforementioned problems in terms of the minimal separators S of G and the components of $G - S$, this enables us to give a scheme for recursive algorithms. In this way, for each of the three problems, we obtain an algorithm that solves the corresponding problem for all graphs G in time $O(n^5 r + m + kr^{k+1}(n+m)n\log n)$, where $k = \mathsf{an}(G)$ and r is the number of minimal separators of G. Moreover, the algorithms can be implemented without knowing the asteroidal number or the number of minimal separators of the input graphs in advance. In that case, the algorithms will generate the correct answers, within the stated timebound. This is of importance, since computing the asteroidal number in general is NP-complete [18].

2 Preliminaries

Throughout the paper, let G denote a graph with vertex set V and edge set E. We denote the number of vertices of G by n, the number of edges of G by m, and the size of a maximum clique in G by $\omega(G)$. For a proper subset $W \subset V$, $G - W$ denotes the subgraph of G obtained by removing the vertices of W. For a vertex $x \in V$, we write $G - x$ instead of $G - \{x\}$. For $\emptyset \neq W \subseteq V$, $G[W]$ denotes the subgraph of G induced by the vertices of W. For any set S, we denote by $S^{[2]}$ the set of all subsets of S of cardinality 2. For any set $\mathfrak{S}$ whose elements are sets itself, we use $\bigcup\mathfrak{S}$ to denote $\bigcup_{S\in\mathfrak{S}} S$. For a vertex $x \in V$, $N(x)$ is the neighborhood of x and $N[x] = \{x\} \cup N(x)$ is the closed neighborhood of x. We say that a sequence $P = (u_0, u_1, \dots, u_l)$ of pairwise distinct vertices of G is a u,v-path of G if $u = u_0$, $v = u_l$, and for $i = 1, \dots, l$ there is an edge $\{u_{i-1}, u_i\} \in E$.

Definition 1. *A subset $A \subseteq V$ is called an* asteroidal set *of G if for each $a \in A$ the vertices of $A \setminus \{a\}$ are contained in one component of $G - N[a]$. The maximum cardinality of an asteroidal set of G is denoted by* $\mathsf{an}(G)$*, and is called the* asteroidal number *of G.*

By definition the vertices of an asteroidal set are pairwise nonadjacent. Hence $\mathsf{an}(G) \leq \alpha(G)$, where $\alpha(G)$ denotes the maximum cardinality of an independent set in G. Furthermore for every k there exist graphs of asteroidal number k, e.g., $\mathsf{an}(C_{2k}) = k$ for $k \geq 2$, where C_n is the chordless cycle on n vertices. Notice that every subset of an asteroidal set is itself asteroidal.

An asteroidal set of cardinality three was called an asteroidal triple (short AT) in [22], where it was shown that chordal graphs without AT are exactly those that are interval graphs.

There are polynomial time algorithms to compute the asteroidal number for graphs in some special classes like HHD-free graphs (including all chordal graphs), claw-free graphs, circular-arc graphs and circular permutation graphs. However the corresponding decision problem remains NP-complete on triangle-free 3-connected 3-regular planar graphs [18].

Definition 2. *A graph H is* chordal *(or* triangulated*) if it does not contain a chordless cycle of length at least four as an induced subgraph.*

Definition 3. *A* triangulation *of G is a graph H with the same vertex set as G such that H is chordal and G is a subgraph of H. A triangulation H of G is called* minimal *if there is no proper subgraph H' of H which is also a triangulation of G.*

Definition 4. *The* treewidth *of G, denoted by* $\mathsf{tw}(G)$*, is the minimum of* $\omega(H)-1$ *taken over all triangulations H of G.*

Definition 5. *The* minimum fill-in *of G, denoted by* $\mathsf{mfi}(G)$*, is the minimum of* $|E(H) \setminus E|$ *taken over all triangulations H of G.*

Definition 6. *Let t be an integer. A* (vertex) t-ranking *of G is a coloring* $c : V \to \{1,\ldots,t\}$ *such that for every pair of vertices x and y with $c(x) = c(y)$ and for every path between x and y there is a vertex z on this path with $c(z) > c(x)$. The* vertex ranking number *of G, denoted by* $\chi_{\mathrm{r}}(G)$*, is the smallest value t for which the graph G admits a t-ranking.*

A proper subset $S \subseteq V$ is a *separator* of G if $G - S$ is disconnected.

Definition 7. *A vertex set $S \subset V$ is an* a, b-separator *of G if the removal of S separates a and b in distinct components of $G - S$. If no proper subset of an a, b-separator S is an a, b-separator then S is a* minimal a, b-separator*. A vertex set $S \subset V$ is a* minimal separator *of G if there exist nonadjacent vertices a and b of G such that S is a minimal a, b-separator of G.*

We define $\mathsf{Comp}(G) = \{X : \varnothing \neq X \subseteq V$ and $G[X]$ is a component of $G\}$. By $\mathsf{Sep}(G)$ we denote the set of all minimal separators of G. The following lemma is well-known and was rediscovered many times (see, e.g., [14]).

Definition 8. *Let S be a separator of G. A component H of $G - S$ is* full *(w.r.t. S) if every vertex of S has at least one neighbor in H.*

Lemma 1. *A set S of vertices of G is a minimal separator of G if and only if $G - S$ has at least two full components.*

Notice that Lemma 1 enables the design of a linear time algorithm that decides whether a given vertex set S is a minimal separator of a given graph G.

Dirac established the following characterization of chordal graphs [13].

Theorem 1. *G is a chordal graph if and only if every minimal separator of G is a clique.*

Definition 9. *Let $\mathfrak{S}$ be any set of vertex subsets of G. Then* $G_{\mathfrak{S}} = (V, E \cup \bigcup_{S \in \mathfrak{S}} S^{[2]})$ *is the graph obtained from G by adding exactly those edges, which are not present in G and which are edges of a complete graph on some $S \in \mathfrak{S}$.*

Now we can state a characterization of minimal triangulations.

Theorem 2. *A graph H is a minimal triangulation of G if and only if* $H = G_{\mathsf{Sep}(H)}$.

In the following lemma we mention two useful characteristics of minimal triangulations (see e.g. [19]).

Lemma 2. *If H is a minimal triangulation of a graph G then*

1. *If a and b are nonadjacent in H, then every minimal a, b-separator in H is also a minimal a, b-separator in G.*
2. *If S is a minimal separator in H and if C is the vertex set of a component of $H - S$, then C induces also a component in $G - S$.*

3 Recurrence Relations and minimal separators

Some well-known graph parameters can be computed by applying recurrence relations involving the set of all minimal separators of the graph under consideration. The most prominent examples concern the treewidth, minimum fill-in and vertex ranking, and appeared in [19], [19], and [12] respectively. In the next theorem, $G(\{S\}, C) = G_{\{S\}}[S \cup C]$ and $\mathsf{fill}(S) = \binom{|S|}{2} - |E(G[S])|$.

Theorem 3. *Let G be a graph which is not complete. Then*

$$\mathsf{tw}(G) = \min_{S \in \mathsf{Sep}(G)} \max_{C \in \mathsf{Comp}(G-S)} \mathsf{tw}(G(\{S\}, C)).$$

$$\mathsf{mfi}(G) = \min_{S \in \mathsf{Sep}(G)} \Big(\mathsf{fill}(S) + \sum_{C \in \mathsf{Comp}(G-S)} \big(\mathsf{mfi}(G(\{S\}, C)) - \mathsf{fill}(S) \big) \Big).$$

$$\chi_{\mathrm{r}}(G) = \min_{S \in \mathsf{Sep}(G)} \big(|S| + \max_{C \in \mathsf{Comp}(G-S)} \chi_{\mathrm{r}}(G[C]) \big).$$

Besides many efficient algorithms on special graph classes for the three problems, one has obtained algorithms for AT-free graphs, that are based on the abovementioned recurrence relations, in [19] and [20].

Our major goal in the remainder of this paper is to generalize the approach for AT-free graphs to obtain a general scheme for designing recursive algorithms on graphs which is applicable as soon as there is a recurrence relation for computing the graph parameter under consideration similar to those in Theorem 3. Because of space limitation, we omit all proofs in this extended abstract.

4 Blocks

Blocking sets and blocks are central concepts for the recursive algorithms and the corresponding decompositions.

Definition 10. *A set $\mathfrak{S}$ of minimal separators of G is a* blocking set *if the elements of $\mathfrak{S}$ (except for possibly the empty set) are incomparable with respect to set inclusion and for all $S \in \mathfrak{S}$ the vertex set $\bigcup \mathfrak{S} \setminus S$ is contained in one component of $G - S$.*

Note that in particular any minimal separator of G is a blocking set.

Definition 11. *Let $\mathfrak{S}$ be a blocking set of G with $|\mathfrak{S}| \geq 2$. Then a vertex $v \in V \setminus \bigcup \mathfrak{S}$ is said to be* in the interior of $\mathfrak{S}$ *if, for every $S \in \mathfrak{S}$, the vertex v and the vertex set $\bigcup \mathfrak{S} \setminus S$ are contained in one component of the subgraph $G - S$.*

Lemma 3. *For every blocking set $\mathfrak{S}$ of G, $|\mathfrak{S}| \leq \mathsf{an}(G)$.*

Definition 12. *A pair $(\mathfrak{S}, C)$ is a* block *of G if $\mathfrak{S}$ is a blocking set of G, $C \subseteq V$ and one of the following conditions is fulfilled.*

– *$|\mathfrak{S}| \geq 2$ and the set C is the set of all vertices in the interior of $\mathfrak{S}$.*

– *If $\mathfrak{S}$ contains exactly one element S, then C is the vertex set of a component of $G - S$ or $C = \varnothing$.*
– *$\mathfrak{S} = \varnothing$ and C is the vertex set of a component of G.*

The definition and Lemma 3 immediately imply

Observation 1 *The number of different blocks of G is at most*

$$(|\mathsf{Sep}(G)| + 1) \cdot |V| + \sum_{k=2}^{\mathsf{an}(G)} \binom{|\mathsf{Sep}(G)|}{k}.$$

The following definition is motivated by the recurrence relations in Section 3 and Theorems 1 and 2.

Definition 13. *The* realization $G(\mathfrak{S}, C)$ *of a block* $(\mathfrak{S}, C)$ *of G is the graph* $G_{\mathfrak{S}}[C \cup \bigcup \mathfrak{S}]$.

The definition implies that the realization of any block is a connected graph.

5 Decomposing Blocks

We consider a block $(\mathfrak{S}, C)$ of G, its realization $H = G(\mathfrak{S}, C)$ and a minimal separator T of H. Then for an arbitrary component $H[D]$ of $H - T$, the pair $(\{T\}, D)$ is a block of H. Our major goal in this section is to prove a claim stating that any block $(\{T\}, D)$ of H can be described as a block of G in the following sense: For any block $(\{T\}, D)$ of $H = G(\mathfrak{S}, C)$, there is a block $(\mathfrak{T}, D')$ of G such that the corresponding realizations are exactly the same graphs, i.e., $G(\mathfrak{T}, D') = H(\{T\}, D)$.

The consequence is that any algorithm, which recursively computes a minimal separator T for the current graph H and then calls itself on the realization of the block $(\{T\}, D)$ for each component D of $H - T$ until the current graph is complete, will only work on realizations of blocks of the input graph G. Together with Lemma 3 and Observation 1 this implies, that each recursive algorithm of this type checks at most $O(|\mathsf{Sep}(G)|^{\mathsf{an}(G)})$ realizations of blocks of the input graph G.

We start with two lemmas that are essential for this section. First we consider minimal separators of realizations.

Lemma 4. *Let $(\mathfrak{S}, C)$ be a block of G and let a and b be nonadjacent vertices in $G(\mathfrak{S}, C)$. Then every minimal a, b-separator in $G(\mathfrak{S}, C)$ is a minimal a, b-separator in G.*

The next lemma classifies the minimal separators of realizations into three types.

Lemma 5. *Let $(\mathfrak{S}, C)$ be a block of G and let T be a minimal separator of $H = G(\mathfrak{S}, C)$. Then exactly one of the following three conditions holds:*

Type 1: *there are distinct minimal separators $S_1, S_2 \in \mathfrak{S}$ with $T \subset S_1$ and $T \subset S_2$,*

Type 2: *there is exactly one separator $S_0 \in \mathfrak{S}$ such that $T \subset S_0$,*

Type 3: *$T \setminus S \neq \varnothing$ for all $S \in \mathfrak{S}$.*

Furthermore, in Types 1 and 2 the graph $H - T$ has exactly two components.

Proposition 1 (Type 1). *Let $(\mathfrak{S}, C)$ be a block of G and let T be a minimal separator of $H = G(\mathfrak{S}, C)$ such that there exist at least two different minimal separators in $\mathfrak{S}$ containing T. Then $C = \varnothing$, $|\mathfrak{S}| = 2$ and for each $S \in \mathfrak{S}$ we have $H(\{T\}, S \setminus T) = G(\{S\}, \varnothing)$.*

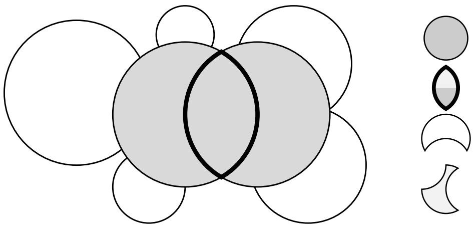
minimal separators in $\mathfrak{S}$
the minimal separator T
components of $G - \bigcup\mathfrak{S}$ with vertices not in the interior of $\mathfrak{S}$
vertices in the interior of $\mathfrak{S}$

Fig. 1. Type 1

Let $(\{S_1, S_2\}, \varnothing)$ be a block of G. By Proposition 1 the unique minimal separator $T = S_1 \cap S_2$ of $G(\{S_1, S_2\}, \varnothing)$ decomposes $(\{S_1, S_2\}, \varnothing)$ into two other blocks of G. We define the decomposition of $(\{S_1, S_2\}, \varnothing)$ by

$$\mathsf{Dec}(\{S_1, S_2\}, \varnothing, T) = \{(\{S_1\}, \varnothing), (\{S_2\}, \varnothing)\}.$$

Proposition 2 (Type 2). *Let $(\mathfrak{S}, C)$ be a block of G and let T be a minimal separator of $H = G(\mathfrak{S}, C)$ such that there is a unique separator $S_0 \in \mathfrak{S}$ with $T \subset S_0$. Let $\mathfrak{T} = \mathfrak{S} \setminus \{S_0\}$ and $D = C \cup \bigcup\mathfrak{T}$. Then $H[D]$ and $H[S_0 \setminus T]$ are the components of $H - T$. Furthermore $(\{T\} \cup \mathfrak{T}, C)$ is a block of G with $G(\{T\} \cup \mathfrak{T}, C) = H(\{T\}, D)$, and $(\{S_0\}, \varnothing)$ is a block of G with $G(\{S_0\}, \varnothing) = H(\{T\}, S_0 \setminus T)$.*

Let $(\mathfrak{S}, C)$ be a block of G and let T be a minimal separator of $H = G(\mathfrak{S}, C)$ such that there is a unique separator $S_0 \in \mathfrak{S}$ with $T \subset S_0$. Based on Proposition 2 we define

$$\mathsf{Dec}(\mathfrak{S}, C, T) = \{(\{S_0\}, \varnothing), (\{T\} \cup \mathfrak{S} \setminus \{S_0\}, C)\}.$$

Proposition 3 (Type 3). *Let $(\mathfrak{S}, C)$ be a block of G and let T be a minimal separator of $H = G(\mathfrak{S}, C)$ such that $T \setminus S \neq \varnothing$ for all $S \in \mathfrak{S}$. Let $H[D]$ be a component of $H - T$. Let $\mathfrak{T} = \{S : S \in \mathfrak{S}$ and $S \setminus T \subseteq D\}$ and $D' = D \setminus \bigcup\mathfrak{T}$. Then $(\{T\} \cup \mathfrak{T}, D')$ is a block of G and $G(\{T\} \cup \mathfrak{T}, D') = H(\{T\}, D)$.*

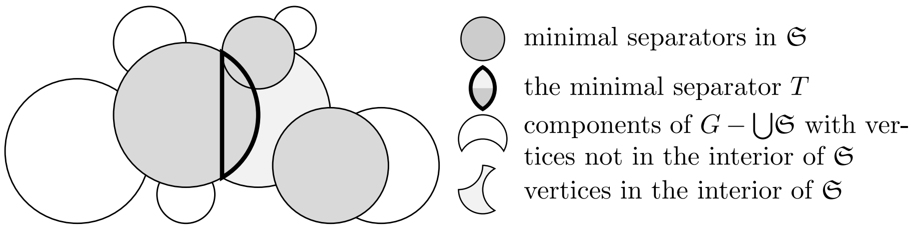
minimal separators in $\mathfrak{S}$
the minimal separator T
components of $G - \bigcup\mathfrak{S}$ with vertices not in the interior of $\mathfrak{S}$
vertices in the interior of $\mathfrak{S}$

Fig. 2. Type 2

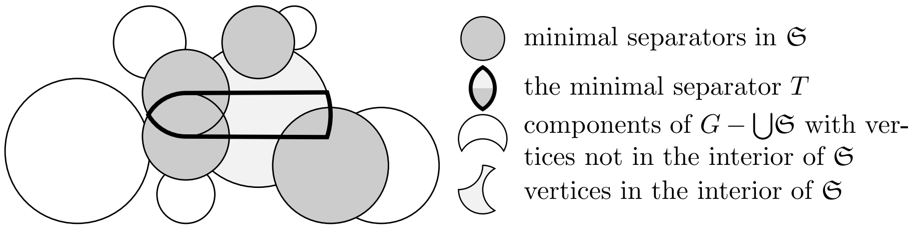
minimal separators in $\mathfrak{S}$
the minimal separator T
components of $G - \bigcup\mathfrak{S}$ with vertices not in the interior of $\mathfrak{S}$
vertices in the interior of $\mathfrak{S}$

Fig. 3. Type 3

Let $(\mathfrak{S}, C)$ be a block of G and let T be a minimal separator of $H = G(\mathfrak{S}, C)$ such that $T \setminus S \neq \varnothing$ for all $S \in \mathfrak{S}$. In this case let $\{H[D_i] : i \in I\}$ be the set of components of $H - T$. Based on Proposition 3 we define

$$\mathsf{Dec}(\mathfrak{S}, C, T) = \{(\{T\} \cup \{S : S \in \mathfrak{S} \text{ and } S \cap D_i \neq \varnothing\}, C \cap D_i) : i \in I\}.$$

The following theorem summarizes Lemma 5 and Propositions 1, 2 and 3.

Theorem 4. *Let $(\mathfrak{S}, C)$ be a block of G and let T be a minimal separator of $H = G(\mathfrak{S}, C)$. Then we have a bijection between the blocks (T, D) corresponding with the components of $H - T$ and the blocks $(\mathfrak{T}, D')$ in $\mathsf{Dec}(\mathfrak{S}, C, T)$ such that $G(\mathfrak{T}, D') = H(\{T\}, D)$.*

6 Algorithms

The approach of the previous section enables two different types of algorithms. One type is a dynamic programming algorithm as used in [8,12,19,20].

Here we use another type of algorithm sometimes called recursive algorithm with memoization (see e.g. [9]). First we describe the generic version. The input is a graph $G = (V, E)$. In a preprocessing the algorithm computes $\mathsf{Sep}(G)$ using the listing algorithm given in [17].

The procedure `compute` is the heart of the algorithm. It is recursive via `access`. The macros `compute` and `main` use `collect`, `complete`, `initialize`, `update` and `start`, which are specific to the algorithmic problem.

The algorithm uses a data structure X that can store any block $(\mathfrak{S}, C)$ of a graph $G = (V, E)$ with a value $p(\mathfrak{S}, C)$, and retrieve these values. Suppose $V =$

```
procedure main;
begin
  compute $\mathsf{Sep}(G)$;
  $p \leftarrow$ start;
  for $C \in \mathsf{Comp}(G)$ do $p \leftarrow$ collect(access($\varnothing, C$));
  return($p$)
end.

procedure access($\mathfrak{S}, C$);
begin
  if not present($\mathfrak{S}, C$) then compute($\mathfrak{S}, C$);
  return(value($\mathfrak{S}, C$))
end;

procedure compute($\mathfrak{S}, C$);
begin
  $p \leftarrow$ complete;
  if $G(\mathfrak{S}, C)$ is not complete then
    for $T \in \mathsf{Sep}(G)$ do
      if $T$ is a minimal separator of $G(\mathfrak{S}, C)$ then
        begin
          $q \leftarrow$ initialize;
          for $(\mathfrak{T}, D) \in \mathsf{Dec}(\mathfrak{S}, C, T)$ do $q \leftarrow$ update(access($\mathfrak{T}, D$));
          $p \leftarrow \min\{p, q\}$;
        end;
  store($\mathfrak{S}, C, p$)
end;
```

$\{1, 2, \ldots, n\}$. Any block $(\mathfrak{S}, C)$ is stored as a set $C \subseteq V$ followed by a sequence of the minimal separators $S_1, S_2, \ldots, S_j$ in $\mathfrak{S}$ that are lexicographically ordered (as subsets of V). The data structure X supports the following operations:

– `store`$(\mathfrak{S}, C, p)$ stores for the block $(\mathfrak{S}, C)$ the value p,
– `present`$(\mathfrak{S}, C)$ returns **true**, if an operation `store`$(\mathfrak{S}, C, p)$ has been performed before, for any value of p, and **false** otherwise, and
– `value`$(\mathfrak{S}, C)$ returns the value p of the (last) operation `store`$(\mathfrak{S}, C, p)$, if `present`$(\mathfrak{S}, C) =$ **true**.

All three operations can be executed by iterated search for a vertex in the universe V. A single search can be done in time $O(\log n)$ by standard techniques. To find a whole block $(\mathfrak{S}, C)$ we need $|C| + |\bigcup \mathfrak{S}|$ single searches if $|\mathfrak{S}| \leq 1$ and $\sum_{S \in \mathfrak{S}} |S|$ single searches if $|\mathfrak{S}| \geq 2$. We refer to [3] for an implementation of a related data structure that can easily be extended to one satisfying our purposes. Notice that our algorithm calls `value`$(\mathfrak{S}, C)$ only if `present`$(\mathfrak{S}, C) =$ **true**. Furthermore if `store`$(\mathfrak{S}, C)$ is called, then `present`$(\mathfrak{S}, C) =$ **false**, i.e., for each block of G, `store` is called at most once.

	treewidth	minimum fill-in	ranking number
`collect`(c)	$\max\{p, c\}$	$p + c$	$\max\{p, c\}$
`complete`	$\lvert C \cup \bigcup \mathfrak{S}\rvert - 1$	$\mathsf{fill}(C \cup \bigcup \mathfrak{S})$	$\lvert C\rvert$
`initialize`	0	$\mathsf{fill}(T)$	$\lvert T \cap C\rvert$
`update`(c)	$\max\{q, c\}$	$q + c - \mathsf{fill}(T)$	$\max\{q, c + \lvert T \cap C\rvert\}$
`start`	0	0	0

We consider the running time of our algorithm on an input graph $G = (V, E)$ with $|V| = n$, $|E| = m$, $|\mathsf{Sep}(G)| = r$ and $\mathsf{an}(G) = k$. First the algorithm in [17] needs $O(n^5 r + m)$ time to list all minimal separators of G.

For the following analysis, we assume that all macros can be evaluated in constant time. (If this is not the case in a particular application, it should be easy to achieve the corresponding time bound with a similar analysis.) To determine the overall running time, we estimate the running time of `compute`$(\mathfrak{S}, C)$ for any block $(\mathfrak{S}, C)$ of G without counting the running time of those recursive calls `compute`$(\mathfrak{T}, D)$ for which `present`$(\mathfrak{T}, D) =$ **false** when `compute`$(\mathfrak{T}, D)$ is called. For any block $(\mathfrak{S}, C)$ of G, `access` calls `compute` at most once, namely when `present`$(\mathfrak{S}, C) =$ **false**. In this case, for each minimal separator T of G, `compute` needs $O(n + m)$ time to test whether T is a minimal separator of $G(\mathfrak{S}, C)$ and, if so, to compute the blocks in $\mathsf{Dec}(\mathfrak{S}, C, T)$. For each of the at most n blocks $(\mathfrak{T}, D)$ in $\mathsf{Dec}(\mathfrak{S}, C, T)$, `access`$(\mathfrak{T}, D)$ is executed. If `access` is called for a block $(\mathfrak{T}, D)$ of G, when `present`$(\mathfrak{T}, D) =$ **true**, then `access` does not call `compute`.

Procedure `access` looks up the value $p(\mathfrak{S}, C)$ in the data structure X. Using an implementation of the data structure X, similar to the one described in [3], one look-up can be done in time $\sum_{S \in \mathfrak{S}} |S| \cdot O(\log n) = O(kn \log n)$.

By Observation 1, the number of different blocks of the input graph G is at most $(r+1)n + \sum_{i=2}^{k} \binom{r}{i}$. Consequently, the total running time of the algorithm is $O(n^5 r + m + kr^{k+1}(n + m)n \log n)$.

Theorem 5. *The generic algorithm runs in $O(n^5 r + m + kr^{k+1}(n + m)n \log n)$ time, where r is the number of minimal separators and k is the asteroidal number of the input graph (under some assumptions on the macros).*

The generic algorithm can be used to compute a graph parameter which can be evaluated via a certain type of recurrence involving the minimal separators of the graph (see e.g. Section 3). In particular, Theorem 5 has the following consequence.

Corollary 1. *For each of the problems* Treewidth, Minimum fill-in *and* Vertex ranking *there is an algorithm to compute the corresponding graph parameter for any input graph G in time $O(n^5 r + m + kr^{k+1}(n + m)n \log n)$, where r is the number of minimal separators of G and $k = \mathsf{an}(G)$.*

References

1. Arnborg, S., Efficient algorithms for combinatorial problems on graphs with bounded decomposability—A survey. *BIT* **25** (1985), pp.2–23.
2. Arnborg, S., D. G. Corneil and A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM J. Alg. Disc. Meth.* **8** (1987), pp. 277–284.
3. Bodlaender, H., Kayles on special classes of graphs - An application of Sprague-Grundy theory. *Proceedings of the 18th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'92*, Springer-Verlag, 1993, LNCS 657, pp. 90–102.
4. Bodlaender, H., A tourist guide through treewidth, *Acta Cybernetica* **11** (1993), pp. 1–23.
5. Bodlaender, H., A linear time algorithm for finding tree-decompositions of small treewidth, *SIAM Journal on Computing* **25** (1996), pp. 1305–1317.
6. Bodlaender, H., J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, Zs. Tuza, Rankings of graphs, *SIAM Journal on Discrete Mathematics* **11** (1998), pp. 168–181
7. Bodlaender, H. L., J. R. Gilbert, H. Hafsteinsson, T. Kloks, Approximating treewidth, pathwidth and minimum elimination tree height, *Journal of Algorithms* **18** (1995), pp. 238–255.
8. Bodlaender, H., T. Kloks and D. Kratsch, Treewidth and pathwidth of permutation graphs, *SIAM Journal on Discrete Mathematics* **8** (1995), pp. 606–616.
9. Cormen, T. H., C. E. Leiserson and R. L. Rivest, *Introduction to algorithms*, MIT Press, Cambridge, Massachusetts, USA, 1990.
10. Corneil, D. G., S. Olariu and L. Stewart, Asteroidal triple-free graphs, *SIAM Journal on Discrete Mathematics.* **10** (1997), pp. 399–430.
11. Corneil, D. G., S. Olariu and L. Stewart, A linear time algorithm to compute dominating pairs in asteroidal triple-free graphs, *Proceedings of ICALP'95*, Springer-Verlag, LNCS 944, 1995, pp. 292–302.
12. Deogun, J. S., T. Kloks, D. Kratsch and H. Müller, On vertex ranking for permutation and other graphs, *11th Annual Symposium on Theoretical Aspects of Computer Science*, Springer-Verlag, LNCS 775, 1994, pp. 747–758.
13. Dirac, G. A., On rigid circuit graphs, *Abh. Math. Sem. Univ. Hamburg* **25** (1961), pp. 71–76.
14. Golumbic, M. C., *Algorithmic graph theory and perfect graphs*, Academic Press, New York, 1980.
15. Habib, M. and R. H. Möhring, Treewidth of cocomparability graphs and a new order-theoretic parameter, *Order* **11** (1994), pp. 47–60.
16. Kloks, T., *Treewidth – Computations and Approximations*, Springer Verlag, LNCS 842, (1994).
17. Kloks, T. and D. Kratsch, Finding all minimal separators of a graph, *SIAM Journal on Computing* **27** (1998), pp. 605-613.
18. Kloks, T., D. Kratsch and H. Müller, Asteroidal sets in graphs, *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'97)*, Springer-Verlag, LNCS 1335 (1997), pp. 229–241.
19. Kloks, T., D. Kratsch and J. Spinrad, On treewidth and minimum fill-in of asteroidal triple-free graphs, *Theoretical Computer Science* **175** (1997), pp. 309-335.
20. Kloks, T., H. Müller and C. K. Wong, Vertex ranking of asteroidal triple-free graphs, *Proceedings of the 7th International Symposium on Algorithms and Computation*, pp. 174–182, Springer Verlag, LNCS 1178, 1996.

21. Leiserson, C. E., Area efficient graph layouts for VLSI, *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science*, 1980, pp. 270–281.
22. Lekkerkerker, C. G. and J. Ch. Boland, Representation of a finite graph by a set of intervals on the real line, *Fundamenta Mathematicae* **51** (1962), pp. 45–64.
23. Lin, I. J., T. A. McKee and D. B. West, Leafage of chordal graphs, Manuscript 1994.
24. Liu, J. W. H., The role of elimination trees in sparse factorization, *SIAM Journal of Matrix Analysis and Applications* **11** (1990), pp. 134–172.
25. Möhring, R. H., Triangulating graphs without asteroidal triples, *Discrete Applied Mathematics* **64** (1996), pp. 281–287.
26. Parra, A., Structural and algorithmic aspects of chordal graph embeddings, PhD. thesis, Technische Universität Berlin, 1996.
27. Prisner, E., Representing triangulated graphs in stars, *Abh. Math. Sem. Univ. Hamburg* **62** (1992), pp. 29–41.
28. Rose, D. J., R. E. Tarjan and G. S. Lueker, Algorithmic aspects of vertex elimination on graphs, *SIAM Journal on Computing* **5** (1976), pp. 266–283.
29. Walter, J. R., Representations of chordal graphs as subtrees of a tree, *Journal of Graph Theory* **2** (1978), pp. 265–267.
30. Yannakakis, M., Computing the minimum fill-in is NP-complete, *SIAM Journal on Algebraic and Discrete Methods* **2** (1981), pp. 77–79.