Exponential time algorithms for the minimum dominating set problem on some graph classes

SERGE GASPERS
University of Bergen, Norway
and
DIETER KRATSCH
Université Paul Verlaine - Metz, France
and
MATHIEU LIEDLOFF and IOAN TODINCA
Université d'Orléans, France

The Minimum Dominating Set problem remains NP-hard when restricted to any of the following graph classes: c-dense graphs, chordal graphs, 4-chordal graphs, weakly chordal graphs and circle graphs. Developing and using a general approach, for each of these graph classes we present an exponential time algorithm solving the Minimum Dominating Set problem faster than the best known algorithm for general graphs. Our algorithms have the following running time: $O(1.4124^n)$ for chordal graphs, $O(1.4776^n)$ for weakly chordal graphs, $O(1.4845^n)$ for 4-chordal graphs, $O(1.4887^n)$ for circle graphs, and $O(1.2273^{(1+\sqrt{1-2c})n})$ for c-dense graphs.

Categories and Subject Descriptors: F.2.2 [**Analysis of algorithms and problem complexity**]: Nonnumerical algorithms and problems

General Terms: Algorithms

Additional Key Words and Phrases: Moderately exponential-time algorithms, dominating set problem, graph classes

1. INTRODUCTION

During the last years there has been a growing interest in the design of exact exponential time algorithms. Various surveys on exact exponential time algorithms have been published recently [Fomin et al. 2005b; Iwama 2004; Schöning 2005; Woeginger 2003; 2004]. In Woeginger's seminal survey [2003], fundamental techniques to design and analyze exact exponential time algorithms are presented. Among others, Fomin et al. present treewidth based algorithms [2005b].

Author's address: S. Gaspers, Department of Informatics, University of Bergen, N-5020 Bergen, Norway, `serge.gaspers@ii.uib.no`.
D. Kratsch, Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France, `kratsch@univ-metz.fr`.
M. Liedloff and I. Todinca, Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, 45067 Orléans Cedex 2, France `(Mathieu.Liedloff|Ioan.Todinca)@univ-orleans.fr`.

Exponential time algorithms for special graph classes have been studied in particular for graphs of maximum degree three and for planar graphs (see e.g. [Fomin et al. 2005b]).

Known results. A set $D \subseteq V$ of a graph $G = (V, E)$ is dominating if every vertex of $V \setminus D$ has at least one neighbor in D. Given a graph $G = (V, E)$, the Minimum Dominating Set problem (`MDS`) asks to compute a dominating set of minimum cardinality.

Exact exponential time algorithms for the Minimum Dominating Set problem have not been studied until recently. By now there is a large interest in this particular problem. In 2004 three papers with exact algorithms for `MDS` have been published. In [2004] Fomin et al. presented an $O(1.9379^n)$ time algorithm for general graphs and algorithms for split graphs, bipartite graphs and graphs of maximum degree three with running time $O(1.4143^n)$, $O(1.7321^n)$, $O(1.5144^n)$, respectively. Exact algorithms for `MDS` on general graphs have also been given by Randerath and Schiermeyer [2004] and by Grandoni [2006]. Their running times are $O(1.8899^n)$ and $O(1.8026^n)$, respectively.

These algorithms have been significantly improved by Fomin et al. in [2005a] where the authors obtain exact algorithms for `MDS` on general graphs. Their simple Branch & Reduce algorithm is analyzed using the so-called Measure & Conquer approach, and the upper bounds on the worst case running times are established by the use of non standard measures. Their algorithm has running time $O(1.5263^n)$ and needs polynomial space. Using memorization one can speed up the running time to $O(1.5137^n)$ needing exponential space then. Both variants are based on algorithms for the Minimum Set Cover problem where the input consists of a universe $\mathcal{U}$ and a collection $\mathcal{S}$ of subsets of $\mathcal{U}$, and the problem requires to find a minimum number of subsets in $\mathcal{S}$ such that their union is equal to $\mathcal{U}$. These algorithms need running time $O(1.2354^{|\mathcal{U}|+|\mathcal{S}|})$ and polynomial space, or running time $O(1.2303^{|\mathcal{U}|+|\mathcal{S}|})$ and exponential space [Fomin et al. 2005a].

Recently these algorithms have been improved by van Rooij and Bodlaender. Thus the currently fastest exact algorithm to compute a minimum set cover has running time $O(1.2273^{|\mathcal{U}|+|\mathcal{S}|})$ and it needs exponential space [van Rooij and Bodlaender 2008]. All running times proved in this paper directly build on this new algorithm.

Fomin and Høie [2006] used a treewidth based approach to establish an algorithm to compute a minimum dominating set for graphs of maximum degree three within running time $O(1.2010^n)$. The best known algorithm for `MDS` on planar graphs has running time $O(2^{3.99\sqrt{n}})$ [Dorn 2006].

It is known that the problem `MDS` is NP-hard when restricted to circle graphs [Keil 1993] and chordal graphs [Booth and Johnson 1982], and thus also for weakly chordal and 4-chordal graphs. The NP-hardness of `MDS` for c-dense graphs is shown in Section 4.

Our results. In this paper we study the Minimum Dominating Set problem for various graph classes and we obtain algorithms with a running time $O(\alpha^n)$ better than the best known algorithm solving `MDS` on general graphs. Here the value of α depends on the graph class. More precisely $\alpha < 1.5$ for all classes except for c-dense graphs. (If $c \geq 0.0202$ then $\alpha < 1.5$ for c-dense graphs.)

Table I. Running time of our algorithms

graph class	running time	
c-dense graphs	$O(1.2273^{n(1+\sqrt{1-2c})})$	(Section 4)
chordal graphs	$O(1.4124^n)$	(Section 5)
circle graphs	$O(1.4887^n)$	(Section 6)
4-chordal graphs	$O(1.4845^n)$	(Section 7)
weakly chordal graphs	$O(1.4776^n)$	(Section 8)

In Section 4 we give an $O(1.2273^{n(1+\sqrt{1-2c})})$ time algorithm for c-dense graphs, i.e. for all graphs with at least cn^2 edges, where c is a constant between 0 and 1/2. In Section 5 we present an exact algorithm solving the MDS problem on chordal graphs in time $O(1.4124^n)$. In Section 6 an $O(1.4887^n)$ time algorithm to solve MDS on circle graphs is presented. In Section 7 an $O(1.4845^n)$ time solving the MDS problem on 4-chordal graphs is given. In Section 8 an $O(1.4776^n)$ time solving the MDS problem on weakly chordal graphs is given.

We are using two general frameworks. "Many vertices of high degree" relies heavily on the minimum set cover algorithm of van Rooij and Bodlaender [2008]. It is applied to c-dense graphs. Our treewidth based approach uses in fact the "many vertices of high degree" approach for graphs of large treewidth, and otherwise it applies the MDS algorithm using a tree decomposition. This approach is applied to chordal, circle, 4-chordal and weakly chordal graphs.

The algorithms for circle, 4-chordal and weakly chordal graphs rely on a linear upper bound of the treewidth in terms of the maximum degree. Such bounds are interesting in their own. A related result for graphs of small chordality is provided in [Bodlaender and Thilikos 1997]. We are not aware of any previous result of this kind for any of the three classes.

2. PRELIMINARIES

Let $G = (V, E)$ be an undirected and simple graph. For a vertex $v \in V$ we denote by $N(v)$ the neighborhood of v and by $N[v] = N(v) \cup \{v\}$ the closed neighborhood of v. For a given subset of vertices $S \subseteq V$, $N(S)$ (resp. $N[S] = N(S) \cup S$) denotes the neighborhood (resp. close neighborhood) of S and $G[S]$ denotes the subgraph of G induced by S. The maximum degree of a graph G is denoted by $\Delta(G)$ or by Δ if there is no ambiguity.

A clique is a set $C \subseteq V$ of pairwise adjacent vertices. The maximum cardinality of a clique in a graph G is denoted by $\omega(G)$. A dominating set D of a graph $G = (V, E)$ is a subset of vertices such that every vertex of $V - D$ has at least one neighbor in D. The minimum cardinality of a dominating set of G is the domination number of G, and it is denoted by $\gamma(G)$.

Major tools used in this paper are tree decompositions and treewidth of graphs. Both notions have been introduced by Robertson and Seymour in [1986]. We recall some useful definitions.

Definition 2.1 *(Tree decomposition).* Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $(\{X_i : i \in I\}, T)$ where each X_i, $i \in I$, is a subset of V and T is a tree with elements of I as nodes such that we have the following properties :

(1) $\cup_{i \in I} X_i = V$;

(2) $\forall \{u, v\} \in E$, $\exists i \in I$ such that $\{u, v\} \subseteq X_i$;

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

The width of a tree decomposition is equal to $\max_{i \in I} |X_i| - 1$.

Definition 2.2 *(Treewidth).* The *treewidth* of a graph G is the minimum width over all its tree decompositions and it is denoted by $tw(G)$. A tree decomposition is called *optimal* if its width is $tw(G)$.

Definition 2.3 *(Nice tree decomposition).* A *nice tree decomposition* $(\{X_i : i \in I\}, T)$ is a tree decomposition satisfying the following properties:

(1) every node of T has at most two children;
(2) If a node i has two children j and k, then $X_i = X_j = X_k$ (i is called a Join Node);
(3) If a node i has one child j, then either
  (a) $|X_i| = |X_j| + 1$ and $X_j \subset X_i$ (i is called an Insert Node);
  (b) $|X_i| = |X_j| - 1$ and $X_i \subset X_j$ (i is called a Forget Node).

LEMMA 2.4 [KLOKS 1994]. *For a constant k, given a tree decomposition of a graph G of width k and N nodes, one can find a nice tree decomposition of G of width k with at most $4N$ nodes in $O(n)$ time, where n is the number of vertices of the graph G.*

Nice tree decompositions of small width are exploited in an algorithm of Alber et al. which is crucial for our work.

THEOREM 2.5 [ALBER ET AL. 2002]. *There is an $4^\ell n^{O(1)}$ time algorithm taking as input a graph $G = (V, E)$ and a tree decomposition T of G of width at most ℓ, which computes a minimum dominating set of G.*

Their algorithm is sketched in the proof of Lemma 5.1 where our algorithm to compute a minimum dominating set in chordal graphs in time $3^\ell n^{O(1)}$ is given.

Structural and algorithmic properties of graph classes will be mentioned in the corresponding sections. For definitions and properties of graph classes not given in this paper we refer to [Brandstädt et al. 1999; Golumbic 1980].

3. GENERAL FRAMEWORK

Our algorithms solve the NP-hard Minimum Dominating Set problem by exploiting two particular properties of the input graph G:

—G has many vertices of high degree:
$|\{v \in V : d(v) \geq t - 2\}| \geq t$ for some (large) positive integer t
(see Theorem 3.1);

—there is a constant $c > 0$ such that $tw(H) \leq c \cdot \Delta(H)$ for all induced subgraphs H of G, and there is an algorithm to compute a tree decomposition of H of width at most $c \cdot \Delta(H)$ in polynomial time[1]
(see Theorem 3.3).

[1]In fact running time $4^{c \cdot \Delta(H)} n^{O(1)}$ suffices.

We describe methods using and combining those properties to establish exponential time algorithms solving MDS for a variety of graph classes for which the problem remains NP-hard. The designed algorithms are faster than the best known algorithm for general graphs.

3.1 Many vertices of high degree

The following theorem shows that graphs with sufficiently many vertices of high degree allow to speed up any $O(\alpha^{2n})$ time algorithm solving MDS for general graphs which is based on an algorithm for Minimum Set Cover of running time $O(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$. This is the case for the currently best known algorithm solving MDS which is based on an $O(1.2273^{|\mathcal{U}|+|\mathcal{S}|})$ algorithm for Minimum Set Cover [van Rooij and Bodlaender 2008], i.e. $\alpha = 1.2273$.

Theorem 3.1. *Suppose there is a $O(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ algorithm computing a minimum set cover of any input $(\mathcal{U}, \mathcal{S})$. Let $t(n) : \mathbb{N} \to \mathbb{R}^+$. Then there is a $O(\alpha^{2n-t(n)})$ time algorithm to solve the* MDS *problem for all input graphs G fulfilling $|\{v \in V : d(v) \geq t(n) - 2\}| \geq t(n)$, where n is the number of vertices of G.*

Proof. Let $G = (V, E)$ be a graph fulfilling the conditions of the theorem and let $t = t(n) \geq 0$. Let $T = \{v \in V : d(v) \geq t - 2\}$; thus $|T| \geq t$. Notice that for each minimum dominating set D of G either at least one vertex of T belongs to D, or $T \cap D = \emptyset$.

This allows to find a minimum dominating set of G by the following branching in two types of subproblems: "$v \in D$" for each $v \in T$, and "$T \cap D = \emptyset$". Thus we branch into $|T| + 1$ subproblems and for each subproblem we shall apply the $O(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ time minimum set cover algorithm to solve the subproblems. Recall the transformation given in [Fomin et al. 2005a]: the minimum set cover instance corresponding to the instance G for the MDS problem has universe $\mathcal{U} = V$ and $\mathcal{S} = \{N[u] : u \in V\}$, and thus $|\mathcal{U}| + |\mathcal{S}| = 2n$. Consequently the running time for a subproblem will be $O(\alpha^{2n-x})$, where x is the number of vertices plus the number of subsets eliminated from the original minimum set cover problem for the graph G.

Now let us consider the two types of subproblems. For each vertex $v \in T$, we choose v in the minimum dominating set and we execute the $O(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ time Minimum Set Cover algorithm on an instance of size at most $2n - (d(v) + 1) - 1 \leq 2n - t$. Indeed, we remove from the universe $\mathcal{U}$ the elements of $N[v]$ and we remove from $\mathcal{S}$ the set corresponding to v. When branching into the case "discard T" we have an instance of set cover of size at most $2n - |T| = 2n - t$ since for every $v \in T$ we remove from $\mathcal{S}$ the set corresponding to each v. $\square$

Corollary 3.2. *Let $t(n) : \mathbb{N} \to \mathbb{R}^+$. Then there is a $O(1.2273^{2n-t(n)})$ time algorithm to solve the* MDS *problem for all input graphs G fulfilling $|\{v \in V : d(v) \geq t(n) - 2\}| \geq t(n)$, where n is the number of vertices of G.*

3.2 Treewidth based approach

For a survey on treewidth based exponential time algorithms we refer to [Fomin et al. 2005b].

The following theorem shows how to solve the MDS problem on a class of graphs fulfilling the condition $tw(G) \leq c\Delta(G)$ for all graphs of the class, where c is a fixed constant. The idea is that such graphs either have many vertices of high degree or their maximum degree is small and thus their treewidth is small. In the first case the algorithm of the previous subsection is used. In the second case the $4^{tw(G)}n^{O(1)}$ time algorithm to solve the `MDS` problem of Alber et al. [2002] (Theorem 2.5) is used. To balance the running time of the two parts, a parameter λ is appropriately chosen.

Algorithm DS-HighDeg-SmallTw(a graph $G = (V, E)$)
Input : A graph G fulfilling the conditions of Theorem 3.3.
Output: The domination number $\gamma(G)$ of G.

```
$\lambda \leftarrow \lambda(c, \alpha)$  /* the value of $\lambda$ is given in Theorem 3.3   */
$X \leftarrow \{u \in V : d(u) \geq \lambda n/c\}$
if $|X| \geq \lambda n/c$ then
    use the algorithm of Theorem 3.1 and return the result
else
    use the algorithm of Theorem 2.5 and return the result
```

THEOREM 3.3. *Suppose there is a $O(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ algorithm computing a minimum set cover of any input $(\mathcal{U}, \mathcal{S})$. Let $c > 0$ be a constant. Let $\mathcal{G}$ be a hereditary class of graphs such that $tw(G) \leq c \cdot \Delta(G)$ for all $G \in \mathcal{G}$. Furthermore, suppose that there is an algorithm that for any input graph $G \in \mathcal{G}$ computes a tree decomposition of width at most $c \cdot \Delta(G)$ in polynomial time.*
Then there is a $\max(\alpha^{2n-\lambda n/c}, 4^{(c+1)\lambda n/c})n^{O(1)}$ time algorithm to solve the `MDS` *problem for all input graphs of $\mathcal{G}$, where $\lambda = \lambda(c, \alpha) = \frac{2}{\frac{1+d}{c}+d}$ and $d = 1/\log_4(\alpha)$.*

PROOF. The algorithm first constructs the vertex set X containing all vertices having a degree larger than $\lambda n/c$ (see algorithm **DS-HighDeg-SmallTw**).

By definition, for all $v \in X$, $d(v) > \lambda n/c$. Thus, if $|X| \geq \lambda n/c$, then we apply the algorithm of Theorem 3.1, and thus a minimum dominating set can be found in time $\alpha^{2n-\lambda n/c}n^{O(1)}$.

Otherwise $|X| < \lambda n/c$ and $\Delta(G - X) < \lambda n/c$. Note that $G - X$ belongs to the hereditary graph class $\mathcal{G}$ since it is an induced subgraph of G. Therefore $tw(G - X) \leq c\Delta(G - X)$. It follows that $tw(G - X) < c\lambda n/c = \lambda n$. As adding one vertex to a graph increases its treewidth at most by one, $tw(G) \leq tw(G - X) + |X| < \lambda n + \lambda n/c = (c + 1)\lambda n/c$. Now our algorithm computes a tree decomposition of width at most $(c + 1)\lambda n/c$ in polynomial time, and then it uses the algorithm of Theorem 2.5. Thus in this case a minimum dominating set can be found in time $4^{tw(G)}n^{O(1)} \leq 4^{(c+1)\lambda n/c}n^{O(1)}$.

As a consequence, a minimum dominating set of the graph G can be found in time $\max(\alpha^{2n-\lambda n/c}, 4^{(c+1)\lambda n/c})n^{O(1)}$. □

COROLLARY 3.4. *Under the assumptions of Theorem 3.3, there is an algorithm of running time $\max(1.2273^{2n-\lambda n/c}, 4^{(c+1)\lambda n/c})n^{O(1)}$ to solve the* `MDS` *problem for all input graphs G fulfilling $tw(G) \leq c\Delta(G)$, where c is a constant, $\lambda = \lambda(c) = \frac{2}{\frac{1+d}{c}+d}$ and $d = 1/\log_4(1.2273)$.*

Table II. Running time of the algorithm in Corollary 3.4 for some values of c

value of c	1.5	2	2.5
running time	$O(1.4723^n)$	$O(1.4776^n)$	$O(1.4815^n)$
value of c	3	4	5
running time	$O(1.4845^n)$	$O(1.4887^n)$	$O(1.4916^n)$

As we have shown, both methods can be adapted to speed up the algorithms by using any faster Minimum Set Cover algorithms established by future work.

In the rest of the paper we show how the abovementioned general methods can be applied to dense graphs (Section 4), chordal graphs (Section 5), circle graphs (Section 5), 4-chordal graphs (Section 7) and weakly chordal graphs (Section 8). We provide also some algorithms giving a tree decomposition of width at most $c\Delta(G)$, where c is a constant depending of the considered graph class.

4. DENSE GRAPHS

It is known that problems like Independent Set, Hamiltonian Circuit and Hamiltonian Path remain NP-complete when restricted to graphs having a large number of edges [Schiermeyer 1995]. In this section we first show that Dominating Set also remains NP-complete for c-dense graphs. Then we present an exponential time algorithm for the `MDS` problem on this graph class. The algorithm uses the "many vertices of high degree" approach of the previous section.

Definition 4.1. A graph $G = (V, E)$ is said to be *c-dense* (or simply dense if there is no ambiguity), if $|E| \geq cn^2$ where c is a constant with $0 < c < 1/2$.

An easy way to show that an NP-complete graph problem remains NP-complete for c-dense graphs, for any c with $0 < c < 1/2$, is to construct a graph G' by adding a sufficiently large complete graph as new component to the original graph G such that G' is c-dense. This simple reduction can be used to show that various NP-complete graph problems remain NP-complete for c-dense graphs. To name a few problems: Independent Set (since $\alpha(G') = \alpha(G) + 1$), Partition into Cliques (since $\kappa(G') = \kappa(G) + 1$), Vertex Cover, Feedback Vertex Set and Minimum Fill-In.

In this way it can be shown that Dominating Set is NP-complete for c-dense graphs by a polynomial-time many-one reduction from the NP-complete problem Dominating Set for split graphs.

Theorem 4.2. *For any constant c with $0 < c < 1/2$, the problem to decide, whether a c-dense graph has a dominating set of size at most k is NP-complete, even when the inputs are restricted to split graphs.*

The main idea of our algorithm is to find a large subset of vertices of large degree.

Lemma 4.3. *For some fixed $1 \leq t \leq n$, $1 \leq t' \leq n-1$, any graph $G = (V, E)$ with $|E| \geq 1 + \dfrac{(t-1)(n-1) + (n-t+1)(t'-1)}{2}$ has a subset $T \subseteq V$ such that*

(i) $|T| \geq t$,

(ii) for every $v \in T$, $d(v) \geq t'$.

Proof. Let $1 \leq t \leq n$, $1 \leq t' \leq n-1$, and a graph $G = (V, E)$ such that there is no subset T with the previous properties. Then for any subset $T \subseteq V$ of size at least t, there exists a vertex $v \in T$ such that $d(v) < t'$. Then such a graph can only have at most $k = k_1 + k_2$ edges where : $k_1 = (t-1)(n-1)/2$ which corresponds to $t-1$ vertices of degree $n-1$ and $k_2 = (n-t+1)(t'-1)/2$ which corresponds to $n-(t-1)$ vertices of degree $t'-1$. Observe that if one of the $n-(t-1)$ vertices has a degree greater than $t'-1$ then the graph has a subset T with the required properties, a contradiction. □

Lemma 4.4. *Every c-dense graph $G = (V, E)$ has a set $T \subseteq V$ fulfilling*

(i) $|T| \geq \left\lfloor n - \dfrac{\sqrt{9-4n+4n^2-8cn^2}-3}{2} \right\rfloor$,

(ii) for every $v \in T$, $d(v) \geq \left\lfloor n - \dfrac{\sqrt{9-4n+4n^2-8cn^2}-3}{2} \right\rfloor - 2$.

Proof. We apply Lemma 4.3 with $t' = t-2$. Since we have a dense graph, $|E| \geq cn^2$. Using inequality $1+((t-1)(n-1)+(n-t+1)(t-3))/2 \leq cn^2$ we obtain that in a dense graph the value of t in Lemma 4.3 is such that $n + \frac{3-\sqrt{9-4n+4n^2-8cn^2}}{2} \geq t$. □

Using the "many vertices of high degree" approach we establish

Theorem 4.5. *For any c with $0 < c < 1/2$, there is a $O(1.2273^{(1+\sqrt{1-2c})n})$ time algorithm to solve the* `MDS` *problem on c-dense graphs.*

Proof. Combining Theorem 3.1, Corollary 3.2 and Lemma 4.4 we obtain an algorithm for solving the Minimum Dominating Set problem in time

$$1.2273^{2n - \left\lfloor n - \frac{\sqrt{9-4n+4n^2-8cn^2}-3}{2} \right\rfloor} = O(1.2273^{n(1+\sqrt{1-2c})}).$$

□

5. CHORDAL GRAPHS

In this section we present an exponential time algorithm for the Minimum Dominating Set problem on chordal graphs. We use a treewidth based approach with a faster algorithm for solving MDS on chordal graphs using clique trees.

A graph is *chordal* if it has no chordless cycle of length greater than three. Chordal graphs are a well-known graph class with its own chapter in Golumbic's monograph [1980]. Split graphs, strongly chordal graphs and undirected path graphs are well-studied subclasses of chordal graphs.

We shall use the clique tree representation of chordal graphs that we view as a tree decomposition of the graph. A tree T is as *clique tree* of a chordal graph $G = (V, E)$ if there is a bijection between the maximal cliques of G and the nodes of T such that for each vertex $v \in V$ the cliques containing v induce a subtree of T. It is well-known that $tw(G) \geq \omega(G) - 1$ for all graphs. Furthermore the clique tree of a chordal graph G is an optimal tree decomposition of G, i.e. its width is precisely $\omega(G) - 1$.

Lemma 5.1. *There is a $3^{tw(G)} n^{O(1)}$ time algorithm to compute a minimum dominating set on chordal graphs.*

Proof. The algorithm of Alber et al. in Theorem 2.5 uses a nice tree decomposition of the input graph and a standard bottom up dynamic programming on the tree decomposition. The crucial idea is to assign three different "colors" to the vertices of a bag:

—"black", meaning that the vertex belongs to the dominating set,
—"white", meaning that the vertex is already dominated,
—"gray", meaning that the vertex is not yet dominated.

Now let us assume that the input graph is chordal. A clique tree T of G can be computed in linear time [Blair and Peyton 1993]. By Lemma 2.4, a nice optimal tree decomposition of G can be computed from the optimal tree decomposition T in time $O(n)$ and it has at most $4n$ nodes. Since G is chordal every bag in the nice tree decomposition is a clique. Therefore no bag can have both a black vertex and a gray vertex. Due to this restriction there are at most $2^{|X|}$ possible so-called vector colorings of a bag X (instead of $3^{|X|}$ for general graphs).

Consequently the running time of the algorithm of Alber et al. for chordal graphs is $3^{tw(G)}n^{O(1)}$, where the only modification is to use clique trees and to restrict allowed vector colorings of a bag such that black and gray vertices simultaneously are forbidden. $\square$

We use the following Corollary of Theorem 3.1.

Corollary 5.2. *There is an algorithm taking as input a graph G and a clique C of G and solving the* MDS *problem in time $O(1.2273^{2n-|C|})$.*

Proof. Note that every vertex in C has degree at least $|C| - 1$. $\square$

Our algorithm on chordal graphs works as follow: If the graph has large treewidth then it necessarily has a large clique and we apply Corollary 5.2. Otherwise the graph has small treewidth and we use Lemma 5.1.

Theorem 5.3. *There is an $O(1.4124^n)$ time algorithm to solve the* MDS *problem on chordal graphs.*

Proof. If $tw(G) < 0.3142n$, by Lemma 5.1, MDS is solvable in time $O(3^{0.3142n}) = O(1.4124^n)$. Otherwise, $tw(G) \geq 0.3142n$ and using Corollary 5.2 we obtain an $O(1.2273^{2n-0.3142n}) = O(1.4124^n)$ time algorithm. $\square$

6. CIRCLE GRAPHS

In this section, we present an exponential time algorithm for MDS on circle graphs in a treewidth based approach.

A *circle graph* is an intersection graph of chords in a circle. More precisely, G is a circle graph, if there is a circle with a collection of chords, such that one can associate in a one-to-one manner a chord to each vertex of G such that two vertices are adjacent in G if and only if the corresponding chords have a nonempty intersection. The circle and all the chords are called a *circle model* of the graph.

Our algorithm heavily relies on a linear upper bound on the treewidth of circle graphs in terms of the maximum degree: $tw(G) \leq 4\Delta(G)$ for every circle graph G. The principal result of this section is a constructive proof of this inequality.

Our approach is based on the fundamental ideas of Kloks' algorithm to compute the treewidth of circle graphs [Kloks 1996]. We start with a brief summary of this algorithm. Consider the circle model of a circle graph G. Go around the circle and place a new point (a so-called *scanpoint*) between every two consecutive end points of chords. The treewidth of a circle graph can be computed by considering all possible triangulations of the polygon $\mathcal{P}$ formed by the convex hull of these scanpoints. The weight of a triangle in this triangulation is the number of chords in the circle model that cross this triangle. The weight of triangulation $\mathcal{T}$ is the maximum weight of a triangle in $\mathcal{T}$. The treewidth of the graph is the minimum weight minus one over all triangulations $\mathcal{T}$ of $\mathcal{P}$. To find an optimal tree decomposition of G, the algorithm in [Kloks 1996] uses dynamic programming to compute a minimum weight triangulation of $\mathcal{P}$.

THEOREM 6.1 [KLOKS 1996]. *There exists an $O(n^3)$ algorithm to compute the treewidth of circle graphs, that also computes an optimal tree decomposition.*

We rely on the following technical definitions in our construction of a tree decomposition of width at most $4\Delta(G)$ for each circle graph G. The construction will be given in the proof of Theorem 6.5.

Definition 6.2. A *scanline* $\tilde{s} = \langle \tilde{a}, \tilde{b} \rangle$ is a chord connecting two scanpoints $\tilde{a}$ and $\tilde{b}$.

To avoid confusion, we call *vertex chords* the chords of the circle model that represent the vertices of the corresponding circle graph. Scanlines are chords as defined above and the general term *chord* refers to both scanlines and vertex chords. To emphasize the difference between scanlines and vertex chords we use different notations: A vertex chord v connecting two end points c and d in the circle model of the graph is denoted $v = [c, d]$. This notation is also used if we consider chords in general. We adapt the standard convention that two vertex chords never intersect on the circle. Moreover, we say that two scanlines with empty intersection or intersecting in exactly one point on the circle (a scanpoint) are *non-crossing*.

Definition 6.3. Let c_1 and c_2 be two non-crossing chords. A chord c is *between* c_1 and c_2 if every path from an end point of c_1 to an end point of c_2 along the circle passes through an end point of c.

Definition 6.4. A set C of chords is *parallel* if and only if

(i) the chords of C are non-crossing, and
(ii) if $|C| > 2$, then for every subset of three chords in C, one of these chords is between the other two.

We refer to Figure 1 for examples of sets of chords that are parallel and non parallel.

A set S of scanlines is *maximal parallel* if there exists no vertex chord v such that $S \cup \{v\}$ is parallel. Given a maximal parallel set of scanlines S, consider the maximal size subpolygons of $\mathcal{P}$ that do not properly intersect any scanline of S (but there may be scanlines of S on their boundaries). For such a subpolygon of $\mathcal{P}$, either one or two edges are scanlines of S. We say that these polygons are *delimited* by one or two scanlines of S and we call *outer polygon $\mathcal{P}_{\tilde{s}}$ with respect to S* such

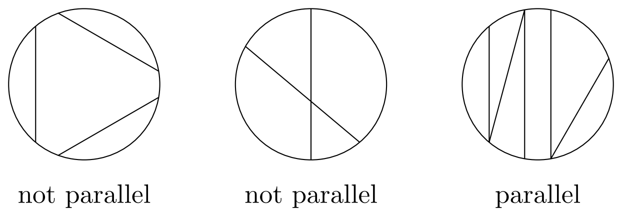

Fig. 1. Examples of parallel and non parallel sets of chords

a polygon delimited by one scanline $\tilde{s} \in S$ and *inner polygon* $\mathcal{P}_{\tilde{s}_1,\tilde{s}_2}$ *with respect to* S such a polygon that is delimited by two scanlines $\tilde{s}_1, \tilde{s}_2 \in S$ and contains at least one end point of a vertex chord (otherwise, it is already triangulated). The inner and outer polygons are defined with respect to a set of maximal parallel set of scanlines S, but we allow ourselves to not state this explicitely if it is clear from the context.

The following theorem is one of the main results of this paper. It shows that the treewidth $tw(G)$ of circle graphs can be upper bounded by a linear function of the maximum degree $\Delta(G)$ of the graph G.

The idea for the proof is to construct an algorithm that computes a triangulation of $\mathcal{P}$ (the triangulation is not necessarily optimal) and to prove that each triangle of this triangulation has weight at most $4\Delta(G)$. Before presenting the algorithm in detail, let us mention some of its major ideas. The algorithm separates $\mathcal{P}$ into "slices" by scanning some appropriately chosen vertex chords in the circle model of the graph, where a vertex chord v is scanned by adding two sharp triangles to the partly constructed triangulation: two scanlines parallel to v and one scanline crossing v to form two triangles. The slices are made thinner and thinner by adding scanlines to the partly constructed triangulation until no slice can be cut into a pair of slices by scanning a vertex chord any more, and this procedure gives a maximal parallel set of scanlines for the slice. When triangulating the "middle part" of any slice, we use the property that no vertex chord is parallel to the two scanlines delimiting the slice to show that the algorithm will not create triangles with a weight exceeding $4\Delta(G)$. The borders of the slices are triangulated recursively by first separating them into slices (in the perpendicular orientation of the previous slices) by scanning some chords and processing the resulting slices similarly.

The most interesting procedure of our algorithm is **TriangInner**, which is also crucial for our upper bound $4\Delta(G)$.

Theorem 6.5. *For every circle graph G, $tw(G) \leq 4\Delta(G)$.*

Proof. The theorem clearly holds for edgeless graphs. Let G be a circle graph with at least one edge and $\mathcal{P}$ be the polygon as previously described. We construct a triangulation of $\mathcal{P}$ such that every triangle has weight at most 4Δ, that is it intersects at most 4Δ vertex chords, and therefore the corresponding tree decomposition has width at most $4\Delta - 1$.

Note that by the definition of a circle graph, every vertex chord intersects at most Δ other vertex chords. The triangulation of the polygon $\mathcal{P}$ is obtained by constructing the corresponding set of scanlines S which is explained by the following procedures. Along with the description of our algorithm, we also analyze the number of vertex chords that cross each triangle and show that it is less than or equal to 4Δ.

We say that a procedure is valid if it does not create triangles with weight higher than 4Δ and if it does not create crossing scanlines.

```
Algorithm TriangCircle(circle model of a graph $G$)
Input   : A circle model of a graph $G$.
Output: A triangulation of weight at most $4\Delta(G)$ of the polygon defined by the
        scanpoints of this circle model.
  Choose any vertex chord $v$ in the circle model of $G$;
  $S \leftarrow$ ScanChord($\emptyset, v$);
  return ParaCuts(S);
```

The validity of Algorithm **TriangCircle** depends on the validity of the procedures **ScanChord** and **ParaCuts**. Note that no scanline crosses v, which is a condition for **ScanChord**. Moreover **ScanChord** produces a parallel set of scanlines, which is a condition for **ParaCuts**.

```
ScanChord($S, v = [a, b]$)
Input   : A set of scanlines $S$ and a vertex chord $v = [a, b]$ such that no scanline
        of $S$ crosses $v$.
Output: A set of scanlines triangulating the polygon defined by the neighboring
        scanpoints of the end points of $v$.
  Let $\tilde{c}$ and $\tilde{c}'$ (respectively $\tilde{d}$ and $\tilde{d}'$) be the two scanpoints closest to $a$
  (respectively $b$) such that the order of the points on the circle is $\tilde{c}, a, \tilde{c}', \tilde{d}', b, \tilde{d}$;
  Let $\tilde{s}_1 = \langle \tilde{c}, \tilde{d} \rangle, \tilde{s}_2 = \langle \tilde{c}', \tilde{d}' \rangle$ and $\tilde{s}_3 = \langle \tilde{c}, \tilde{d}' \rangle$;
  if $\tilde{c} = \tilde{d}$ (or $\tilde{c}' = \tilde{d}'$) then
    $X \leftarrow \{\tilde{s}_2\}$ (or $\{\tilde{s}_1\}$);
  else
    $X \leftarrow \{\tilde{s}_1, \tilde{s}_2, \tilde{s}_3\}$;
  return $X$;
```

The procedure **ScanChord** returns a set X of one or three scanlines. They form at most two triangles: $\tilde{c}, \tilde{d}, \tilde{d}'$ and $\tilde{c}, \tilde{d}', \tilde{c}'$. Each of them intersects at most $\Delta + 1$ vertex chords: v and the vertex chords crossing v. Furthermore, at most Δ vertex chords cross $\tilde{s}_1$ and $\tilde{s}_2$, precisely the vertex chords that cross v. The scanlines of X do not intersect any scanline of S as any scanline intersecting a scanline of X intersects also v.

In the procedure **ParaCuts**, the notions of inner and outer polygons are used with respect to S (see Figure 2). In the while-loop, the chosen vertex chord v does not cross a scanline of S since $S \cup \{v\}$ is required to be parallel. Thus, when the procedure **ScanChord** is called, its conditions are satisfied. After the while-loop,

```
**ParaCuts**($S$)
Input  : A set of parallel scanlines $S$.
Output : A triangulation of weight at most $4\Delta(G)$ of the polygon defined by the
         scanpoints of the circle model.

  while $S$ is not maximal parallel do
      Choose a vertex chord $v$ such that $S \cup \{v\}$ is parallel;
      $S \leftarrow S \cup \textbf{ScanChord}(S, v)$;
  Let $\tilde{s}_1$ and $\tilde{s}_2$ be the scanlines delimiting the two outer polygons;
  $S \leftarrow S \cup \textbf{TriangOuter}(S, \tilde{s}_1) \cup \textbf{TriangOuter}(S, \tilde{s}_2)$;
  foreach inner polygon $P_{\tilde{t}_1,\tilde{t}_2}$ do
      $S \leftarrow S \cup \textbf{TriangInner}(S, \tilde{t}_1, \tilde{t}_2)$
  return $S$
```

S is maximal parallel. Every vertex chord intersecting an outer polygon crosses therefore the scanline delimiting this outer polygon, and there is no vertex chord between two scanlines delimiting an inner polygon, which are necessary conditions for **TriangOuter** and **TriangInner**. Moreover, at most Δ vertex chords cross each of the delimiting scanlines and no scanline of S intersects the inner and outer polygons.

```
**TriangOuter**($S, \tilde{s} = \langle \tilde{a}, \tilde{b} \rangle$)
Input  : A set of scanlines $S$ and a scanline $\tilde{s} \in S$ satisfying the following
         conditions:
         (i) every vertex chord intersecting $\mathcal{P}_{\tilde{s}}$ crosses $\tilde{s}$,
         (ii) at most $2\Delta$ vertex chords cross $\tilde{s}$, and
         (iii) no scanline of $S$ intersects $\mathcal{P}_{\tilde{s}}$.
Output : A set of scanlines triangulating the outer polygon $\mathcal{P}_{\tilde{s}}$.

  $X \leftarrow \emptyset$;
  foreach scanpoint $\tilde{p}_i \in \mathcal{P}_{\tilde{s}} \setminus \{\tilde{a}, \tilde{b}\}$ do
      $X \leftarrow X \cup \{\langle \tilde{a}, \tilde{p}_i \rangle\}$
  return $X$
```

In the procedure **TriangOuter**, at most 2Δ vertex chords intersect the outer polygon $\mathcal{P}_{\tilde{s}}$. So, any triangulation of $\mathcal{P}_{\tilde{s}}$ produces triangles with weight at most 2Δ. As the procedure produces a triangulation of $\mathcal{P}_{\tilde{s}}$, it is valid.

Consider the input of the procedure **TriangInner**. There are at most 3Δ vertex chords inside the quadrilateral $\tilde{a}_1, \tilde{b}_1, \tilde{b}_2, \tilde{a}_2$ since there is no vertex chord crossing both the lines $\tilde{a}_1, \tilde{a}_2$ and $\tilde{b}_1, \tilde{b}_2$ (there is no vertex chord between $\tilde{s}_1$ and $\tilde{s}_2$). As fewer vertex chords cross $\tilde{a}_1, \tilde{a}_2$ than $\tilde{b}_1, \tilde{b}_2$, at most $3\Delta/2$ vertex chords cross the new scanline $\tilde{t} = \langle \tilde{a}_1, \tilde{a}_2 \rangle$. So, when **OuterParaCuts**$(S, \tilde{t})$ is called, the condition that $\tilde{t}$ intersects at most 2Δ vertex chords is respected. For every end point e_i of a vertex chord v_i that crosses $\tilde{s}_1$, two triangles are created: $\tilde{a}_1, \tilde{d}_{i-1}, \tilde{d}_i$ and $\tilde{d}_i, \tilde{d}_{i-1}, \tilde{d}'_i$.

The following claim is both the bottleneck and the crucial point of our argument.

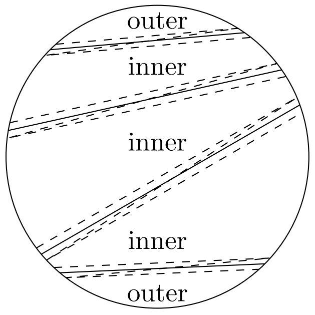

Fig. 2. **ParaCuts**

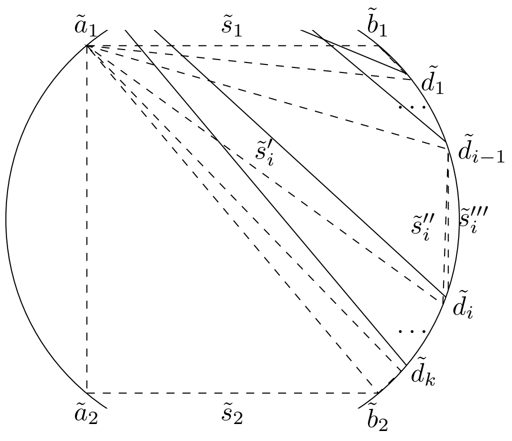

Fig. 3. **TriangInner**

```
**TriangInner**$(S, \tilde{s}_1 = \langle \tilde{a}_1, \tilde{b}_1 \rangle, \tilde{s}_2 = \langle \tilde{a}_2, \tilde{b}_2 \rangle)$
**Input**  : A set of scanlines $S$ and two scanlines $\tilde{s}_1, \tilde{s}_2 \in S$ satisfying the
             following conditions:
             (i) there is no vertex chord between $\tilde{s}_1$ and $\tilde{s}_2$,
             (ii) at most $\Delta$ vertex chords cross one of $\tilde{s}_1$ and $\tilde{s}_2$, say $\tilde{s}_2$,
             (iii) at most $2\Delta$ vertex chords cross the other scanline, $\tilde{s}_1$, and
             (iv) no scanline of $S$ intersects the inner polygon $\mathcal{P}_{\tilde{s}_1,\tilde{s}_2}$.
**Output**: A set of scanlines triangulating $\mathcal{P}_{\tilde{s}_1,\tilde{s}_2}$.

  Let the end points of $\tilde{s}_1$ and $\tilde{s}_2$ be ordered $\tilde{a}_1, \tilde{b}_1, \tilde{b}_2, \tilde{a}_2$ around the circle.
  Assume w.l.o.g., that fewer vertex chords cross the line $\tilde{a}_1, \tilde{a}_2$ than the line
  $\tilde{b}_1, \tilde{b}_2$;
  Let $\tilde{t} = \langle \tilde{a}_1, \tilde{a}_2 \rangle$;
  $X \leftarrow \{\tilde{t}\}$;
  $X \leftarrow X \cup$ **OuterParaCuts**$(X, \tilde{t})$;
  Go around the circle from $\tilde{b}_1$ to $\tilde{b}_2$ (without passing through $\tilde{a}_1$ and $\tilde{a}_2$).
  Denote by $e_1, ..., e_k$ the encountered end points of those vertex chords that
  cross $\tilde{s}_1$;
  **foreach** $e_i, i = 1..k$ **do**
      Let $\tilde{s}'_i = \langle \tilde{a}_1, \tilde{d}_i \rangle$ with $\tilde{d}_i$ being the scanpoint just after $e_i$;
      Let $\tilde{s}''_i = \langle \tilde{d}_i, \tilde{d}_{i-1} \rangle$ with $\tilde{d}_0 = \tilde{b}_1$;
      Let $\tilde{s}'''_i = \langle \tilde{d}_{i-1}, \tilde{d}'_i \rangle$ with $\tilde{d}'_i$ being the scanpoint just before $\tilde{d}_i$;
      $X \leftarrow X \cup \{\tilde{s}'_i, \tilde{s}''_i, \tilde{s}'''_i\}$;
      $X \leftarrow X \cup$ **OuterParaCuts**$(X, \tilde{s}'''_i)$;
  Let $\tilde{s}_3 = \langle \tilde{d}_k, \tilde{b}_2 \rangle$ and $\tilde{s}_4 = \langle \tilde{b}_2, \tilde{a}_1 \rangle$;
  $X \leftarrow X \cup \{\tilde{s}_3, \tilde{s}_4\}$;
  $X \leftarrow X \cup$ **OuterParaCuts**$(X, \tilde{s}_3)$;
  **return** $X$
```

Claim 6.6. *The triangle $\tilde{a}_1, \tilde{d}_{i-1}, \tilde{d}_i$ intersects at most 4Δ vertex chords.*

Proof. Observe that every vertex chord intersecting this triangle and not crossing $\tilde{s}_1$ crosses either v_i or v_{i-1}. As at most 2Δ vertex chords cross $\tilde{s}_1$, at most Δ cross v_i and at most Δ cross v_{i-1}, the weight of this triangle is at most 4Δ. $\square$

Moreover, at most $2\Delta + 1$ vertex chords cross $\tilde{s}''_i$ and at most 2Δ vertex chords cross $\tilde{s}'''_i$. So, the weight of the triangle $\tilde{d}_i, \tilde{d}_{i-1}, \tilde{d}'_i$ is at most $2\Delta + 1$ and when **OuterParaCuts**$(S, \tilde{s}'''_i)$ is called, the condition that the second parameter of the procedure is a scanline that crosses at most 2Δ vertex chords is respected.
After adding the scanlines $\tilde{s}_3$ and $\tilde{s}_4$ we obtain two more triangles: $\tilde{a}_1, \tilde{d}_k, \tilde{b}_2$ and $\tilde{a}_1, \tilde{b}_2, \tilde{a}_2$. The first one intersects at most 4Δ vertex chords: at most 2Δ cross $\tilde{s}_1$, at most Δ cross v_k and at most Δ cross $\tilde{s}_2$. At most 3Δ vertex chords intersect the triangle $\tilde{a}_1, \tilde{b}_2, \tilde{a}_2$: at most 2Δ intersect $\tilde{s}_1$ and at most Δ intersect $\tilde{s}_2$. Moreover at most 2Δ vertex chords cross $\tilde{s}_3$. So, the conditions of **OuterParaCuts**$(S, \tilde{s}_3)$ are respected.

```
OuterParaCuts($S, \tilde{s} = \langle \tilde{a}, \tilde{b} \rangle$)
Input  : A set of scanlines $S$ and a scanline $\tilde{s} \in S$ such that
         (i) at most $2\Delta$ vertex chords cross $\tilde{s}$, and
         (ii) no scanline of $S$ intersects $\mathcal{P}_{\tilde{s}}$.
Output: A set of scanlines triangulating the outer polygon $\mathcal{P}_{\tilde{s}}$.
  $X \leftarrow \{\tilde{s}\}$;
  while $X$ is not a maximal parallel in $\mathcal{P}_{\tilde{s}}$ do
      Choose a chord $v \in \mathcal{P}_{\tilde{s}}$ such that $X \cup \{v\}$ is parallel;
      $X \leftarrow X \cup$ ScanChord$(X, v)$;
  Let $\tilde{t}$ be the scanline delimiting the recently obtained outer polygon with
  respect to $X$ that is a subpolygon of $\mathcal{P}_{\tilde{s}}$;
  $X \leftarrow X \cup$ TriangOuter$(X, \tilde{t})$;
  foreach inner polygon $P_{\tilde{t}_1,\tilde{t}_2}$ in $\mathcal{P}_{\tilde{s}}$ do
      $X \leftarrow X \cup$ TriangInner$(X, \tilde{t}_1, \tilde{t}_2)$;
  return $X$
```

The procedure **OuterParaCuts** is similar to **ParaCuts** on the outer polygon delimited by $\tilde{s}$. A new set of scanlines $X \leftarrow \{\tilde{s}\}$ is created and is made maximal parallel by calling **ScanChord**. If $\{\tilde{s}\}$ is already maximal parallel, then **TriangOuter**$(X, \tilde{s})$ is called and the conditions of that procedure are respected. If other scanlines had to be added to X to make it maximal parallel, the procedure **TriangOuter**$(X, \tilde{t})$ is called for the outer polygon where $\tilde{t}$ is a scanline of X intersecting at most Δ vertex chords. Moreover, the procedure **TriangInner**$(X, \tilde{t}_1, \tilde{t}_2)$ is called for the inner polygons. Every scanline delimiting the inner polygons intersects at most Δ vertex chords, except $\tilde{s}$ that can intersect up to 2Δ vertex chords. So, we respect the condition for **TriangInner** that one scanline intersects at most Δ vertex chords and the other one at most 2Δ.

We have provided a recursive algorithm to triangulate the polygon $\mathcal{P}$ and have shown that the obtained triangulation does not contain triangles intersecting more than 4Δ vertex chords. Thus the corresponding tree decomposition of G has width at most $4\Delta - 1$. □

Now we apply our treewidth based approach of Section 3 to circle graphs. By the above theorem, for every circle graph G, $tw(G) \leq 4\Delta(G)$. Furthermore the class of circle graphs is hereditary and there is a polynomial time algorithm to compute an optimal tree decomposition of circle graphs (Theorem 6.1). Consequently Theorem 3.3 and Corollary 3.4 can be applied and we obtain

THEOREM 6.7. *There is an algorithm to compute for circle graphs a minimum dominating set in time $O(1.4887^n)$.*

7. 4-CHORDAL GRAPHS

The *chordality* of a graph is the size of its longest chordless cycle. A graph is *4-chordal* if its chordality is at most 4. Thus 4-chordal graphs are a superclass of chordal graphs. We show in this section that, for any 4-chordal graph G, its treewidth is at most $3\Delta(G)$.

A vertex set $S \subset V$ is a *separator* if $G - S$ is disconnected. Given two vertices u and v, S is a *u, v-separator* if u and v belong to different connected components of $G - S$, and S is then said to *separate* u and v. A u, v-separator S is *minimal* if no proper subset of S separates u and v. A vertex set S is a *minimal separator* of G if there exist two vertices u and v in G such that S is a minimal u, v-separator. A connected component C of $G - S$ such that $N(C) = S$ is called a *full component associated to S*.

LEMMA 7.1 [GOLUMBIC 1980]. *A set of vertices S of a graph G is a minimal separator of G if and only if there are two distinct full components associated to S.*

We first show that any minimal separator of a 4-chordal graph G has at most $2\Delta(G)$ vertices. We start with some properties of minimal separators in arbitrary graphs.

LEMMA 7.2 [BERRY ET AL. 2000]. *Let $X \subseteq V(G)$ be a set of vertices inducing a connected subgraph and let C be a connected component of $G - N[X]$. Then $N(C)$ is a minimal separator of G.*

Given a minimal separator S of a graph G, the following lemma provides a technique for computing new minimal separators "close to" S.

LEMMA 7.3 [KLOKS AND KRATSCH 1998]. *Let S be a minimal separator of an arbitrary graph G. Consider a vertex $x \in S$ and a connected component D of $G - (S \cup N(x))$. Then $T = N(D)$ is a minimal separator of G.*

LEMMA 7.4. *Let S be a minimal separator of the 4-chordal graph G. Consider a vertex $x \in S$ and a connected component D of $G - (S \cup N(x))$. If the minimal separator $T = N(D)$ is not a subset of S, then T is of size at most $2\Delta(G)$.*

PROOF. T is a minimal separator of G by Lemma 7.3. Let C be the component of $G - S$ containing D.

Partition T into $T_C = T \cap C$ and $T_S = T \cap S$. We claim that for any $y \in T_S \setminus N(x)$ and for any $z \in T_C$, y and z are adjacent. By contradiction, let P_D be a chordless path from y to z whose internal vertices are contained in D, and P_F a chordless path from x to y whose internal vertices are in a full component F associated to S, different from C. Note that $x - P_F - y - P_D - z - x$ form a chordless cycle of G. Also P_F has at least one internal vertex, thus this cycle is of length at least 5 – a contradiction.

By the previous claim, all the vertices of $T_S \setminus N(x)$ are adjacent to some vertex $z \in T \cap C$. Consequently $|T| \leq |N(x)| + |N(z)| \leq 2\Delta(G)$. $\square$

THEOREM 7.5. *For any minimal separator S of a 4-chordal graph G, we have $|S| \leq 2\Delta(G)$.*

PROOF. Berry et al. have shown in [Berry et al. 2000] that the following process produces all minimal separators of a graph.

During the initialization step compute, for each vertex x, the minimal separators of type $N(C)$, for each connected component C of $G - N[x]$ (see Lemma 7.2).

Then we repeat the following main loop, until no new minimal separators are added: for each minimal separator S obtained during the previous iteration (or during the initialization, if we are in the first iteration), for each vertex $x \in S$ and each component D of $G - (S \cup N(x))$, if $T = N(D)$ is not already in the set of computed minimal separators, we add it to this set (see also Lemma 7.3).

Clearly the minimal separators computed in the initialization step are of size at most $2\Delta(G)$. Suppose that all the minimal separators computed before the kth iteration of the main loop are of size at most $2\Delta(G)$. The separators T obtained during this iteration are also of size at most $2\Delta(G)$, by Lemma 7.4.

Since by [Berry et al. 2000], this algorithm generates all minimal separators of G, we conclude that all minimal separators are of size at most $2\Delta(G)$. $\square$

The following generic algorithm takes a graph $G = (V, E)$ together with a vertex subset Z and a component W of $G - Z$ and computes a tree-decomposition of $G[Z \cup W]$ such that one of the bags contains Z. In particular, **MakeDec**$(G, \emptyset, V)$ computes a tree decomposition of G. By an appropriate choice of the bags, we will show that 4-chordal graphs have tree-decompositions of width $3\Delta(G)$.

A very similar approach has been used in [Bodlaender et al. 1995] to show that a tree decomposition of a graph $G = (V, E)$ with treewidth at most $O(k \log n)$, where k is the treewidth of G and $n = |V|$, can be found in polynomial time. For the sake of completeness, we recall that algorithm **MakeDec**(G, Z, W) computes a tree decomposition of $G[Z \cup W]$.

LEMMA 7.6 (SEE ALSO [BODLAENDER ET AL. 1995]). *A tree decomposition of $G[Z \cup W]$ is computed by algorithm **MakeDec**(G, Z, W).*

PROOF. We prove the statement by induction on the recursive structure of the algorithm. The claim is true if the new bag B is exactly $Z \cup W$. Clearly every vertex of $Z \cup W$ appears in an least one bag. Observe that each edge of $G[Z \cup W]$ has both endpoints in B, or both endpoints in $Z_i \cup W_i$ for some component $W_i \subset W$ of $G - B$. Therefore each edge is covered by a bag of the tree decomposition.

It remains to show that for each vertex $v \in Z \cup W$, the bags of T containing v induce a connected subtree. If $v \notin B$, all bags containing v appear in same

```
Algorithm MakeDec($G$, $Z$, $W$)
Input  : An arbitrary graph $G = (V, E)$, a vertex subset $Z$ and a component $W$
         of $G - Z$.
Output: A tree decomposition such that one of the bags contains $Z$.
  Choose $B \subseteq Z \cup W$ such that $Z$ is strictly contained in $B$;
  /* Algorithms to compute B are given in Theorem 7.7 for 4-chordal
     and in Theorem 8.2 for weakly chordal graphs.                      */
  foreach component $W_i$ of $G - B$ s.t. $W_i \subset W$ do
      $Z_i \leftarrow N_G(W_i) \cap B$;
      $T_i \leftarrow$ MakeDec($G, Z_i, W_i$);
  Let $T$ be the tree decomposition obtained as the disjoint union of all $T_i$'s, to
  which we add a node corresponding to the bag $B$, adjacent in each $T_i$ to a bag
  containing $Z_i$;
  return $T$;
```

tree-decomposition T_i, and the claim holds by induction. If $v \in B$, then v appears exactly in the subtrees T_i such that $v \in Z_i$. Since the bag B is adjacent to a bag of T_i containing Z_i, the bags of T containing v induce a connected subtree. □

In the following, we use algorithm **MakeDec**(G, Z, W) such that at each recursive call Z is a minimal separator of G and W is a full component associated to Z.

Theorem 7.7. *For any 4-chordal graph G, $tw(G) \leq 3\Delta(G)$. Moreover, there is a polynomial time algorithm computing, for any 4-chordal input graph G, a tree decomposition of width at most $3\Delta(G)$.*

Proof. The theorem clearly holds for edgeless graphs. Let G be a 4-chordal graph with at least one edge. We construct a tree decomposition using algorithm **MakeDec** and such that all bags are of size at most $3\Delta(G)$. At the initial step **MakeDec**$(G, \emptyset, V)$, let the new bag be $B_0 = N[x_0]$, for some vertex x_0. Clearly $|B_0| \leq 3\Delta(G)$. For each connected component C of $G - B_0$, its neighborhood $S = N(C)$ is a minimal separator, by Lemma 7.2. The algorithm recursively calls **MakeDec**(G, S, C) for each component C of $G - B_0$. We keep as invariant that for each recursive call **MakeDec**(G, Z, W), Z is a minimal separator and W is a full component associated to it.

Now we claim that for such a call **MakeDec**(G, Z, W), we can always construct a new bag B of size at most $3\Delta(G)$. Choose an arbitrary vertex $x \in Z$ and let $B = Z \cup (N(x) \cap W)$. In particular it is of size at most $3\Delta(G)$ by Theorem 7.5. For each component $W' \subset W$ of $G - (Z \cup N(x))$, its neighborhood $Z' = N_G(W)$ is a minimal separator by Lemma 7.3. The recursive calls are of type **MakeDec**(G, Z', W'), where Z' is a minimal separator of G and W' is a full component of $G - Z'$ associated to Z', so the above construction can be iterated.

The tree decomposition can be obtained in $O(nm)$ time, since for each newly created bag B the computation of the components of $G - B$ and their neighborhoods can be performed in linear time. □

Combining Theorem 7.7, Theorem 3.3 and Corollary 3.4 one establishes :

THEOREM 7.8. *There is an $O(1.4845^n)$ algorithm to compute a minimum dominating set for 4-chordal graphs.*

8. WEAKLY CHORDAL GRAPHS

A graph G is *weakly chordal* if both G and its complement are 4-chordal. It is easy to check that chordal graphs are a proper subclass of weakly chordal graphs, which are in turn a proper subclass of 4-chordal graphs. The treewidth of weakly chordal graphs can be computed in polynomial time [Bouchitté and Todinca 2001].

In this section we show that for any weakly chordal graph G its treewidth is at most $2\Delta(G)$.

For weakly chordal graphs, each minimal separator S is contained in the neighborhood of a vertex or of an edge. We will construct a tree decomposition of the graph such that each bag corresponds to the closed neighborhood of a vertex or of an edge.

LEMMA 8.1 [HAYWARD 1997]. *Let G be a weakly chordal graph. For any minimal separator S of G and any full component C of $G - S$ associated to S, there is a vertex $v \in C$ or an edge e of $G[C]$ such that S is contained in the neighborhood of vertex v or of edge e.*

THEOREM 8.2. *For any weakly chordal graph G, $tw(G) \leq 2\Delta(G)$.*

PROOF. The theorem clearly holds for edgeless graphs. Let G be a weakly chordal graph with at least one edge. The construction of the tree decomposition is quite similar to the one of Theorem 7.7, based on algorithm **MakeDec**. We use bags of size at most $2\Delta(G)$. At the first call **MakeDec(**$G, \emptyset, V$**)**, start with a bag $B_0 = N[x]$, for some arbitrary vertex x. For each component connected C of $G - N[x]$, its neighborhood $S = N(C)$ is a minimal separator. We recursively call **MakeDec(**G, S, C**)** for each component C of $G - B_0$. We keep as invariant that for each recursive call **MakeDec(**G, Z, W**)**, Z is a minimal separator and W is a full component associated to it. We show that under these assumptions, the new bag B can be chosen of size at most $2\Delta(G)$.

We consider, like in Lemma 8.1, a vertex $v \in W$ such that $Z \subseteq N(v)$ or an edge e of $G[W]$ such that $Z \subseteq N(e)$. In the former case we choose $N[v]$ as the new bag B, and in the latter we take $B = N[e]$. In both cases, $Z \subset B$ and the size of B is at most $2\Delta(G)$. For each component $W' \subset W$ of $G - B$, its neighborhood $Z' = N_G(W')$ is a minimal separator by Lemma 7.2. Hence we recursively construct a tree decomposition with bags of size at most $2\Delta(G)$. □

Combining Theorem 8.2, Theorem 3.3 and Corollary 3.4 one establishes :

THEOREM 8.3. *There is an $O(1.4776^n)$ algorithm to compute a minimum dominating set for weakly chordal graphs.*

9. CONCLUSIONS

We presented several exponential time algorithms to solve the Minimum Dominating Set problem on graph classes for which this problem remains NP-hard. All these algorithms are faster than the best known algorithm to solve `MDS` on general graphs. We have also shown that any faster algorithm for the Minimum Set Cover problem, i.e. of running time $O(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ with $\alpha < 1.2273$, could immediately be used to speed up all our algorithms.

Besides classes of sparse graphs (as e.g. cubic graphs [Fomin and Høie 2006]) two other graph classes are of interest: split and bipartite graphs. For split graphs, combining ideas of [Fomin et al. 2004] and [van Rooij and Bodlaender 2008] one easily obtains an $O(1.2273^n)$ algorithm. For bipartite graphs, recently Liedloff [2008] established a dynamic programming based algorithm of running time $O(1.4143^n)$.

The "high degree" and the "treewidth based" method of our paper can most likely also be applied to other NP-hard problems for constructing moderately exponential time algorithms when restricted to graph classes with the corresponding properties. One example is the Independent Dominating Set problem (see [Gaspers and Liedloff 2007]).

The bounds on the treewidth in terms of the maximum degree are interesting in their own and it is likely that such bounds for circle graphs, 4-chordal graphs, weakly chordal graphs or other graph classes can be used to construct exponential time algorithms for NP-hard problems on special graph classes in a way similar to our approach for domination.

REFERENCES

Alber, J., Bodlaender, H. L., Fernau, H., Kloks, T., and Niedermeier, R. 2002. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica 33,* 4, 461–493.

Berry, A., Bordat, J., and Cogis, O. 2000. Generating all the minimal separators of a graph. *Int. J. Found. Comput. Sci. 11,* 3, 397–403.

Blair, J. R. S. and Peyton, B. 1993. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computation*. Vol. 56. Springer-Verlag, Berlin, 1–29.

Bodlaender, H. L., Gilbert, J. R., Hafsteinsson, H., and Kloks, T. 1995. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms 18,* 2, 238–255.

Bodlaender, H. L. and Thilikos, D. M. 1997. Treewidth for graphs with small chordality. *Discrete Appl. Math. 79,* 1-3, 45–61.

Booth, K. S. and Johnson, J. H. 1982. Dominating sets in chordal graphs. *SIAM J. Comput. 11,* 1, 191–199.

Bouchitté, V. and Todinca, I. 2001. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput. 31,* 1, 212–232.

Brandstädt, A., Le, V. B., and Spinrad, J. P. 1999. *Graph classes: A survey.* SIAM Monogr. Discrete Math. Appl., Philadelphia, PA, USA.

Dorn, F. 2006. Dynamic programming and fast matrix multiplication. In *Proceedings of the 14th conference on Annual European Symposium (ESA)*. LNCS, vol. 4168. Springer-Verlag, Berlin, 280–291.

Fomin, F. V., Grandoni, F., and Kratsch, D. 2005a. Measure and conquer: Domination - A case study. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*. LNCS, vol. 3580. Springer-Verlag, Berlin, 191–203.

Fomin, F. V., Grandoni, F., and Kratsch, D. 2005b. Some new techniques in design and analysis of exact (exponential) algorithms. *Bull. EATCS 87*, 47–77.

Fomin, F. V. and Høie, K. 2006. Pathwidth of cubic graphs and exact algorithms. *Inf. Process. Lett. 97,* 5, 191–196.

Fomin, F. V., Kratsch, D., and Woeginger, G. J. 2004. Exact (exponential) algorithms for the dominating set problem. In *Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*. LNCS, vol. 3353. Springer-Verlag, Berlin, 245–256.

Gaspers, S. and Liedloff, M. 2007. A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. Tech. Rep. No 344, Department of Informatics, University of Bergen, Norway.

Golumbic, M. C. 1980. *Algorithmic graph theory and perfect graphs.* Academic Press, New York.

Grandoni, F. 2006. A note on the complexity of minimum dominating set. *J. Discrete Algorithms 4,* 2, 209–214.

Hayward, R. B. 1997. Meyniel weakly triangulated graphs I: Co-perfect orderability. *Discrete Appl. Math. 73,* 3, 199–210.

Iwama, K. 2004. Worst-case upper bounds for kSAT. *Bull. EATCS 82*, 61–71.

Keil, J. M. 1993. The complexity of domination problems in circle graphs. *Discrete Appl. Math. 42,* 1, 51–63.

Kloks, T. 1994. *Treewidth: Computations and approximations.* LNCS, vol. 842. Springer-Verlag, Berlin.

Kloks, T. 1996. Treewidth of circle graphs. *Int. J. Found. Comput. Sci. 7,* 2, 111–120.

Kloks, T. and Kratsch, D. 1998. Listing all minimal separators of a graph. *SIAM J. Comput. 27,* 3, 605–613.

Liedloff, M. 2008. Finding a dominating set on bipartite graphs. *Inf. Process. Lett. 107,* 5, 154–157.

Randerath, B. and Schiermeyer, I. 2004. Exact algorithms for minimum dominating set. Tech. Rep. zaik-469, Zentrum für Angewandte Informatik, Köln, Germany.

Robertson, N. and Seymour, P. D. 1986. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms 7,* 3, 309–322.

Schiermeyer, I. 1995. Problems remaining NP-complete for sparse or dense graphs. *Discuss. Math. Graph Theory 15*, 33–41.

Schöning, U. 2005. Algorithmics in exponential time. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS).* Vol. 3404. Springer-Verlag, Berlin, 36–43.

van Rooij, J. M. M. and Bodlaender, H. L. 2008. Design by measure and conquer, a faster exact algorithm for dominating set. In *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science (STACS).* Dagstuhl Seminar Proceedings, vol. 08001. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 657–668.

Woeginger, G. J. 2003. Exact algorithms for NP-hard problems: A survey. In *Proceedings of the 5th International Workshop on Combinatorial Optimization - Eureka, you shrink!* LNCS, vol. 2570. Springer-Verlag, Berlin, 185–207.

Woeginger, G. J. 2004. Space and time complexity of exact algorithms: Some open problems. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC).* LNCS, vol. 3162. Springer-Verlag, Berlin, 281–290.

Received Month Year; revised Month Year; accepted Month Year