An Exact Algorithm for the Minimum Dominating Clique Problem

Dieter Kratsch Mathieu Liedloff*
Laboratoire d'Informatique Théorique et Appliquée
Université Paul Verlaine - Metz
57045 Metz Cedex 01, France
{kratsch|liedloff}@univ-metz.fr

Abstract

A subset of vertices $D \subseteq V$ of a graph $G = (V, E)$ is a dominating clique if D is a dominating set and a clique of G. The existence problem 'Given a graph G, is there a dominating clique in G?' is NP-complete, and thus both the Minimum and the Maximum Dominating Clique problem are NP-hard. We present an $O(1.3387^n)$ time and polynomial space algorithm that for an input graph on n vertices either computes a minimum dominating clique or reports that the graph has no dominating clique. The algorithm uses the Branch & Reduce paradigm and its time analysis is based on the Measure & Conquer approach. We also establish a lower bound of $\Omega(1.2599^n)$ for the worst case running time of the algorithm. Finally using memorization we obtain an $O(1.3234^n)$ time and exponential space algorithm for the same problem.

Keywords. graph algorithms, exponential time algorithms, dominating clique, dominating set

1 Introduction

In the last decades of the 20th century the research on moderately exponential time algorithms for NP-hard problems had been concentrating on satisfiability problems (see e.g. [15, 23]). Nevertheless, some interesting moderately exponential time algorithms for graph problems had also been established. In the last years the design and analysis of exact exponential time algorithms for NP-hard problems has gone through an exciting growth of interest. Various NP-hard graph problems have attracted attention. For some of them, such as Independent Set, Coloring and Hamiltonian Circuit, exact exponential time algorithms had been studied since a long time [22, 24, 18, 14]. Other problems, such as Dominating Set, Treewidth

*Corresponding author. Phone: +33 3 87 31 54 44 fax: +33 3 87 31 53 09.

and Feedback Vertex Set, have not been considered under this perspective until very recently [4, 25, 21].

The growing interest in moderately exponential time algorithms for NP-hard problems has led to various surveys on exact exponential time algorithms that had been published in the last years [5, 15, 23, 27, 28]. In Woeginger's seminal paper fundamental techniques to design and analyse exact exponential time algorithms are presented [27]. Fomin et al. present various techniques for the design and analysis of Branch & Reduce algorithms, among them Measure & Conquer, Lower Bounds and Memorization [5].

In this paper we study the NP-hard Minimum Dominating Clique problem (abbr. `MinDC`). Our main result is a Branch & Reduce algorithm solving the `MinDC` problem in time $O(1.3387^n)$ and polynomial space.

Basic Definitions. Let $G = (V, E)$ be an undirected and simple graph, i.e. without loops and multiple edges. We denote by n the number of vertices of G. The open neighborhood of a vertex v is denoted by $N(v) = \{u \in V : \{u, v\} \in E\}$, and the closed neighborhood of v is denoted by $N[v] = N(v) \cup \{v\}$. The degree of a vertex v is $|N(v)|$. The subgraph of G induced by $S \subseteq V$ is denoted by $G[S]$. We will write $G - S$ for $G[V - S]$. A set $S \subseteq V$ of vertices is a *clique*, if every pair of its vertices is adjacent; S is *independent* if every pair of its vertices is non adjacent.

For a vertex set $X \subseteq V$, we define $N[X] = \bigcup_{v \in X} N[v]$ and $N(X) = N[X] - X$. For a vertex set $S \subseteq V$, we also define $N_S(v) = N(v) \cap S$ and $N_S[v] = N[v] \cap S$. The S-degree of v, denoted by $d_S(v)$, is $|N_S(v)|$. Similarly, given two subsets of vertices $S \subseteq V$ and $X \subseteq V$, we define $N_S(X) = N(X) \cap S$. We will write $\overline{N_S[v]}$ for $S \setminus N_S[v]$, and $\overline{N_S(v)}$ for $\overline{N_S[v]} \cup \{v\}$.

Given a graph $G = (V, E)$ and two vertices u and v of G, the distance between u and v, denoted by $d(u, v)$, is the length of a shortest path between these two vertices. The *eccentricity* of a vertex v of G is defined as $ecc(v) = \max_{w \in V} d(v, w)$. Finally the *diameter* of G is defined as $diam(G) = max_{u,v \in V} d(u, v)$.

For more information on graphs we refer to [26].

Domination. Let $G = (V, E)$ be a graph. A set $D \subseteq V$ with $N[D] = V$ is called a *dominating set* of G; in other words, every vertex in G either belongs to D or has a neighbor in D. The aim of the Minimum Dominating Set problem is to find a dominating set of minimum cardinality. It is one of the fundamental and well-studied NP-hard graph problems [8]. For a large and comprehensive survey on domination theory, we refer the reader to the books [12, 13] by Haynes, Hedetniemi and Slater. The dominating set problem is also one of the basic problems in parameterized complexity. It is known to be W[2]-complete [3]; and thus unlikely to be fixed parameter tractable. Various variants of the Dominating Set problem have been studied extensively. Dominating sets D of a graph G are often required to have particular additional properties, as e.g. $G[D]$ has no isolates, $G[D]$ is

connected, D is an independent set of G, D is a clique of G, etc. For such particular types of dominating sets the related problem asking to find such a particular dominating set of minimum cardinality is typically NP-hard (see e.g. [13]).

Dominating Cliques. A subset of vertices $D \subseteq V$ of a graph $G = (V, E)$ is a *dominating clique* if D is a dominating set and a clique. The study of dominating cliques was initiated in [1] with a motivation from social sciences. The existence problem 'Given a graph G, decide whether G has a dominating clique' (abbr. `ExDC`) is NP-complete even when restricted to weakly triangulated graphs or when restricted to cocomparability graphs [17]. This implies that both natural optimization versions of the problem are NP-hard. The Minimum Dominating Clique problem (abbr. `MinDC`) asks either to find a dominating clique of minimum cardinality of the input graph G, or to output that G has no dominating clique. The Maximum Dominating Clique problem (abbr. `MaxDC`) asks to find a dominating clique of maximum cardinality, or to output that there is none. The complexity of `MinDC` on various graph classes has been studied in the eighties and nineties (see [17]).

Related Results. Exact exponential time algorithms for the Minimum Dominating Set problem have been studied in a sequence of papers [7, 20, 11, 4]. The fastest known algorithms today are due to Fomin et al. and they are based on a Branch & Reduce algorithm for the Minimum Set Cover problem. Their running time is $O(1.5263^n)$ using polynomial space and $O(1.5137^n)$ using exponential space [4]. The analysis is based on Measure & Conquer and the exponential space algorithm is obtained using memorization.

The above results and the power of the Branch & Reduce paradigm when combined with an analysis using Measure & Conquer, encouraged the study of exact exponential time algorithms for variants of the domination problem. For example, Gaspers and Liedloff established an $O(1.3575^n)$ time Branch & Reduce algorithm to compute a minimum independent dominating set [10]. Furthermore Fomin et al. established an $O(1.9407^n)$ time Branch & Reduce algorithm to compute a minimum connected dominating set [6].

Prior to our work, the only published exact exponential time algorithm solving a dominating clique problem had been established by Culberson et al. [2]. Their algorithm solves the existence problem `ExDC`. Their Branch & Reduce algorithm is used in experimental studies and no analysis of the worst case running time is attempted. We show that the worst case running time of Culberson et al's algorithm is $\Omega(1.4142^n)$.

There is also a $3^{n/3}\, n^{O(1)} = O(1.4423^n)$ time algorithm for `ExDC` or `MaxDC` enumerating all maximal cliques and verifying each for being a dominating set. It uses a polynomial delay algorithm to generate all maximal independent sets [16], and its running time follows from Moon and Moser's result [19] that the number of maximal independent sets of a graph is at most $3^{n/3}$.

Our Results. We present an $O(1.3387^n)$ time and polynomial space algorithm computing a minimum dominating clique of the input graph G, or reporting that G has no dominating clique. Thus our algorithm also solves the existence problem `ExDC`. It is designed using the Branch & Reduce paradigm aiming for simple reduction and branching rules. To analyse the worst case running time of the algorithm we use a non standard measure for the size of an input of a (sub)problem and we rely heavily on the Measure & Conquer technique. The theoretical analysis will be described in full detail. The numerical solution of a system of about 210 linear recurrences each one depending on up to 7 weights is unpractical without a computer. We use a program based on random local search to establish an upper bound of $O(1.3387^n)$ for the worst case running time.[1]

Since current tools for the time analysis of Branch & Reduce algorithms (including Measure & Conquer) seem to overestimate the running time of the algorithm, lower bounds for the worst case running time are of interest. We show that the worst case running time of our algorithm is $\Omega(1.2599^n)$.

Using memorization we also establish an $O(1.3234^n)$ time and exponential space algorithm to solve problem `MinDC`.

Organization. In Section 2 we present the Branch & Reduce algorithm solving the problem `MinDC`. In Section 3 we analyze the algorithm using Measure & Conquer and show that its running time is $O(1.3387^n)$. In Section 4 we provide an exponential lower bound for the worst case running time of our Branch & Reduce algorithm. In Section 5 memorization is used to obtain an $O(1.3234^n)$ algorithm needing exponential space which solves the problem `MinDC`. In Section 6 we compare our Branch & Reduce algorithm with the algorithm of Culberson, Gao and Anton. Finally in Section 7 some open problems are mentioned.

2 A Branch & Reduce Algorithm for `MinDC`

In this section we present our algorithm to solve the NP-hard `MinDC` problem for an input graph $G = (V, E)$. We may assume G to be connected, otherwise G cannot have a dominating clique.

Remark. *Any graph with a dominating clique has diameter at most three, and all its vertices have eccentricity at most two. Both conditions might be useful in an implementation of our algorithm; however it seems difficult using them to improve the worst case running time.*

Our algorithm uses four different types of vertices, and thus maintains a partition of V into four subsets:

[1] While determining optimal values of the weights needs computing power; given the recurrences and the values of the weights, it is conceptually easy to check that the solution of the system is correct.

- S is the set of *selected* vertices, i.e. those vertices that have already been chosen for the potential solution (and thus S is a clique);
- D is the set of *discarded* vertices, i.e. those vertices that have already been removed from the (input) graph;
- $A = \bigcap_{s \in S} N(s) \setminus D$ is the set of *available* vertices, i.e. those vertices that might still be added to the potential solution S;
- $F = V \setminus (S \cup A \cup D)$ is the set of *free* vertices. Those vertices still need to be dominated (hence for each free vertex at least one of its available neighbors must be added to S).

Our recursive algorithm `mdc` finds the size of an optimum solution SOL, i.e. a dominating clique of smallest possible cardinality such that the following properties are fulfilled: (i) $S \subseteq SOL$, (ii) $D \cap SOL = \emptyset$, and (iii) $F \cap SOL = \emptyset$. If no such dominating clique exists then `mdc` returns "∞". For a discussion of halting, reduction and branching rules of the Branch & Reduce algorithm we refer to the next section.

To compute a minimum dominating clique of a given graph $G = (V, E)$, $\texttt{mdc}(G, \{v\}, \emptyset, N(v), V \setminus N[v])$ is called for each vertex $v \in N[w]$, where w is a vertex of G of minimum degree (an idea borrowed from [2]).

To observe correctness of this approach, assume that C is a minimum dominating clique of G. Since C is a dominating set of G it must contain a vertex of $N[w]$, say v_0. Thus choosing $S = \{v_0\}$ and $D = \emptyset$ implies that only neighbors of v_0 are still available and all non neighbors of v_0 become free vertices, and still have to be dominated (by adding some available vertices to S). Thus $\texttt{mdc}(G, \{v_0\}, \emptyset, N(v_0), V \setminus N[v_0])$ returns the size of a minimum dominating clique of G. Clearly if G has no dominating clique then for all $v \in N[w]$, $\texttt{mdc}(\{v\}, \emptyset, N(v), V \setminus N[v])$ returns "∞".

Note that with a slight modification our algorithm could also output a minimum dominating clique SOL, in case there is one, instead of the minimum cardinality $|SOL|$.

3 Analysis of the Algorithm

In this section we prove the correctness and analyse the worst case running time of our Branch & Reduce algorithm.

Correctness. Typically the correctness of Branch & Reduce algorithms is easy to show. The algorithm halts (i.e. the subproblem corresponds to a leaf in the search tree) if either there is a free vertex with no available neighbor (H1), and thus there is no dominating clique for the current instance, or there are no free vertices (H2), and thus S is a dominating clique of minimum size for this instance.

Algorithm mdc(G,S,D,A,F)
Input: A graph $G = (V, E)$ and a partition (S, D, A, F) of its vertex set.
Output: The minimum cardinality of a dominating clique of G respecting the partition, if one exists.

if $\exists u \in F$ *s.t.* $d_A(u) = 0$ **then**
  return ∞ **(H1)**
else if $F = \emptyset$ **then**
  return $|S|$ **(H2)**
else if $\exists v \in A$ *s.t.* $d_F(v) = 0$ **then**
  return mdc$(G, S, D \cup \{v\}, A \setminus \{v\}, F)$ **(R1)**
else if $\exists u \in F$ *s.t.* $d_A(u) = 1$ **then**
  let $v \in A$ **be the unique neighbor of** u **in** A
  return mdc$(G, S \cup \{v\}, D \cup \overline{N_A[v]} \cup N_F(v), N_A(v), F \setminus N_F(v))$ **(R2)**
else if $\exists v_1, v_2 \in A$ *s.t.* $N(v_1) \subseteq N(v_2)$ **then**
  return mdc$(G, S, D \cup \{v_1\}, A \setminus \{v_1\}, F)$ **(R3)**
else if $\exists u_1, u_2 \in F$ *s.t.* $N_A(u_1) \subseteq N_A(u_2)$ **then**
  return mdc$(G, S, D \cup \{u_2\}, A, F \setminus \{u_2\})$ **(R4)**
else if $\exists u \in F$ *s.t.* $\big(\forall v' \in N_A(u), d_F(v') = 1$ *and*
  $\exists v \in N_A(u)$ *s.t.* $A \setminus N_A(u) \subseteq N_A(v)\big)$ **then**
  let $v \in N_A(u)$ **be a such vertex fulfilling the condition** **(R5)**
  return mdc$(G, S \cup \{v\}, D \cup (N_A(u) \setminus \{v\}) \cup \{u\}, A \setminus N_A(u), F \setminus \{u\})$
else
  choose $v \in A$ **of maximum** F**-degree**
  if $d_F(v) = 1$ **then**
    let u **be a vertex of** F
    return $\min_{v \in N_A(u)}\{$**mdc**$(G, S \cup \{v\}, D \cup \{u\} \cup \overline{N_A[v]},$
    $A \setminus \overline{N_A(v)}, F \setminus \{u\})\}$ **(B1)**
  else
    return
    $\min\big($**mdc**$(G, S \cup \{v\}, D \cup \overline{N_A[v]} \cup N_F(v), N_A(v), F \setminus N_F(v)),$
    mdc$(G, S, D \cup \{v\}, A \setminus \{v\}, F)\big)$ **(B2)**

Otherwise the algorithm possibly performs some reduction rules on the problem instance, and then it branches using (B1) or (B2) on two or more subproblems, which are solved recursively. In each subproblem the algorithm selects an available vertex and adds it to S and/or discards vertices of G (i.e. adds them to D), and those changes of S and D imply updates on A and F.

We explain now the correctness of the reduction rules. Let (S, D, A, F) be the current partition.

(R1) If an available vertex v has no free neighbor then v can be discarded.

(R2) If $v \in A$ is the unique free neighbor of $u \in F$ then v must be selected (and added to S) to dominate u.

(R3) If $v_1, v_2 \in A$ such that $N(v_1) \subseteq N(v_2)$ then for any dominating clique C containing v_1 there is the dominating clique $C' = (C - \{v_1\}) \cup \{v_2\}$ with $|C'| \leq |C|$. Thus we may safely discard v_1.

(R4) If $u_1, u_2 \in F$ such that $N_A(u_1) \subseteq N_A(u_2)$ we may discard $u_2 \in F$. To see this, notice that any dominating clique respecting the partition $(S, D \cup \{u_2\}, A, F \setminus \{u_2\})$ contains a neighbor $v \in A$ of u_1, and thus also a neighbor of u_2.

(R5) Let $u \in F$ be a free vertex such that all its neighbors $v \in A$ have u as unique free neighbor. If one of these available neighbors, say v_0, is adjacent to all vertices in $A \setminus N_A(u)$, then there exists a minimum dominating clique containing v_0 respecting the partition. To see this, notice that a dominating clique must contain a vertex of $N_A(u)$, that only one of those will be chosen (all others will be discarded immediately by rule (R1)), and that v_0 is the best choice since it does not force removal of any remaining A-vertices (i.e. those in $A \setminus N_A(u)$).

Now we consider the correctness of the branching rules. Let (S, D, A, F) be a partition of the vertices of G when applying a branching rule. Note that the minimum F-degree of an available vertex is at least 1 since any available vertex v with $d_F(v) = 0$ would have been discarded according to reduction rule (R1).

(B1) If all available vertices have exactly one free neighbor, then (B1) chooses any free vertex $u \in F$. Clearly any dominating clique respecting (S, D, A, F) must contain precisely one neighbor $v \in A$ of u. Thus for each neighbor $v \in A$ of u, (B1) branches to a subproblem by selecting v. The minimum cardinality among all dominating cliques obtained is the minimum cardinality of a dominating clique respecting partition (S, D, A, F).

(B2) For an available vertex v of F-degree at least 2 either v is selected or discarded which is trivially correct.

Analysis of the running time. Typically analysing the worst case running time of a Branch & Reduce algorithm is a challenging task and even the best known methods provide only upper bounds.

In order to bound the progress made by our algorithm at each branching step, we apply the Measure & Conquer approach which was introduced in [4] (see also [5]). To each vertex v of G, we assign a weight depending on its number of free neighbors if v is an available vertex, or depending on its number of available neighbors if v is a free vertex. For $i \geq 0$, let $a_i \in [0, 1]$ be the weight of an available vertex v with $d_F(v) = i$. For $i \geq 0$, let $f_i \in [0, 1]$ be the weight of a free vertex v with $d_A(v) = i$.

The following non standard measure on the size of the input of a subproblem is used:

$$\begin{aligned}\mu &= \mu(G,S,D,A,F)\\ &= \sum_{i=0}^{n}\sum_{\substack{v\in A,\\ d_F(v)=i}} a_i + \sum_{i=0}^{n}\sum_{\substack{v\in F,\\ d_A(v)=i}} f_i\end{aligned}$$

Since (S,D,A,F) is a partition of the vertices of $G=(V,E)$, it is easy see that $\mu(G,S,D,A,F) \leq |A\cup F| \leq |V| = n$.

Based on the rules (H1), (R1) and (R2) we set $a_0 = 0$ and $f_0 = f_1 = 0$. To simplify the analysis we put $a_i = a_{\geq 3}$ for all $i \geq 3$, and $f_i = f_{\geq 6}$ for all $i \geq 6$. Thus by $a_{\geq 3} \in [0,1]$ we denote the weight of an available vertex v with $d_F(v) \geq 3$; and by $f_{\geq 6} \in [0,1]$ we denote the weight of a free vertex v with $d_A(v) \geq 6$.

Therefore the measure that we consider in the next can be written as follows:

$$\begin{aligned}\mu &= \mu(G,S,D,A,F)\\ &= \sum_{\substack{v\in A,\\ d_F(v)=1}} a_1 + \sum_{\substack{v\in A,\\ d_F(v)=2}} a_2 + \sum_{\substack{v\in A,\\ d_F(v)\geq 3}} a_{\geq 3}\\ &\quad + \sum_{\substack{v\in F,\\ d_A(v)=2}} f_2 + \sum_{\substack{v\in F,\\ d_A(v)=3}} f_3 + \sum_{\substack{v\in F,\\ d_A(v)=4}} f_4 + \sum_{\substack{v\in F,\\ d_A(v)=5}} f_5 + \sum_{\substack{v\in F,\\ d_A(v)\geq 6}} f_{\geq 6}\end{aligned}$$

By definition of the weights, $0 \leq a_1, a_2, a_{\geq 3}, f_2, f_3, f_4, f_{\geq 5} \leq 1$. To further simplify the analysis we also impose $a_1 \leq a_2 \leq a_{\geq 3}$ and $f_2 \leq f_3 \leq f_4 \leq f_5 \leq f_{\geq 6}$. [2]

The following quantities will be useful:

$$\Delta f_i = \begin{cases} 0 & \text{if } i \geq 7\\ f_i - f_{i-1} & \text{if } 3 \leq i \leq 6\\ f_2 & \text{if } i = 2\end{cases}$$

Let $P[\mu]$ denote the maximum number of subproblems recursively solved by `mdc` to compute a solution on an instance of size μ.

First let us consider the reduction rules. For each of the five reduction rules, when applying it either a vertex is selected or a non empty set of vertices is discarded. Thus the number of consecutive applications of reduction rules to a subproblem (without intermediate branching) is at most n. Since each reduction rule can easily be implemented such that its execution is

[2] Experience based on computations for other choices of the weights indicates that without those restrictions one hardly obtains a better upper bound for the worst case running time. On the other hand, the number of different weights is crucial for the computational part of the analysis and should be as small as possible.

done in polynomial time, the running time of `mdc` on a subproblem (which corresponds to a node of the search tree), except the time consuming by recursive calls, is polynomial. Furthermore, we want to emphasize that due to the choice of the measure no application of a reduction rule to a problem instance will increase its measure.

Let us now consider the more interesting part: the branching rules (B1) and (B2). In the classical analysis of the running time of Branch & Reduce algorithms with a so-called standard measure, i.e. in case of graphs the measure of a subproblem is the number of vertices, the two branching rules would be fairly easy to analyse and one would obtain just a few linear recurrences. Using Measure & Conquer this analysis is more interesting (and can be quite tedious). On the other hand, Measure & Conquer allows relatively simple algorithms with competitive running times since the tricky case analysis that had been part of Branch & Reduce algorithms (see e.g. [24]) is now 'transferred' to the time analysis of the algorithm.

(B1) The algorithm `mdc` chooses a free vertex u and for each of its available neighbors v, it calls itself recursively with v being selected (i.e. v added to S). Recall that (B1) is applied only when all available vertices have F-degree 1. Thus when `mdc` selects an available neighbor v of u, then all other available neighbors of u would decrease their F-degree to 0 and would be discarded and removed from A by reduction rule (R1). Finally we observe that due to reduction rule (R5), each available neighbor of u must be non-adjacent to at least one vertex of $A \setminus N_A(u)$. Consequently for every $d_A(u) \geq 2$, we obtain a recurrence:

- if $d_A(u) = i$ and $2 \leq i \leq 5$,
$$P[\mu] \leq 1 + i \cdot P[\mu - i \cdot a_1 - f_i - a_1] \tag{1}$$
- if $d_A(u) \geq 6$,
$$P[\mu] \leq 1 + d_A(u) \cdot P[\mu - d_A(u) \cdot a_1 - f_{\geq 6} - a_1] \tag{2}$$

(B2) Algorithm `mdc` chooses an available vertex $v \in A$ of maximum F-degree. Since neither (R1) nor (B1) had been applied we may conclude that $d_F(v) \geq 2$. Using (B2) we branch into two subproblems by either selecting v (branch IN) or discarding v (branch OUT).

To obtain subproblem (branch IN), v is removed from A, all free neighbors of v are removed from F and for all $w \in N_A(N_F(v)) \setminus \{v\}$ their F-degree will decrease. Moreover if a vertex $x \in N_A(N_F(v))$ has only one free neighbor, then x will be removed when reduction rule (R1) is applied to subproblem (branch IN). Hence we obtain

$$\Delta_{IN} \geq a_{d_F(v)} + \sum_{u \in N_F(v)} f_{d_A(u)} + \sum_{w \in N_A(N_F(v)) \setminus \{v\}} (a_{d_F(w)} - a_{d_F(w) - |N_F(v) \cap N_F(w)|}) \tag{3}$$

For the subproblem (branch OUT), v is discarded and thus removed from A. Hence the A-degree of all free neighbors u of v decreases by one. Furthermore, when applying (R2) to the subproblem (branch OUT), the free neighbors having only one available neighbor will be removed from F. To dominate such a vertex, the algorithm has to put its only remaining neighbor in S. For a vertex $y \in A$, we denote by $N_{F2}(y) = \{u \in N_F(y) : d_A(u) = 2\}$ the set of free neighbors of v having precisely two available neighbors and one of those is y. Consequently, we obtain

$$\Delta_{OUT} \geq a_{d_F(v)} + \sum_{u \in N_F(v) \setminus N_{F2}(v)} \Delta f_{d_A(u)} + \sum_{u \in N_{F2}(v)} f_{d_A(u)} + \sum_{w \in N_A(N_{F2}(v)) \setminus \{v\}} a_{d_F(w)} \tag{4}$$

Finally using equations (3) and (4) we obtain the recurrence

$$P[\mu] \leq 1 + P[\mu - \Delta_{OUT}] + P[\mu - \Delta_{IN}]. \tag{5}$$

Optimizing the values of the weights. To establish the best possible upper bound of the worst case running time of algorithm `mdc` with the above recurrences we need to choose the weights a_1, a_2, $a_{\geq 3}$, f_2, f_3, f_4, f_5 and $f_{\geq 6}$ as to minimize the solution $O(\alpha^n)$ of the system of recurrences (1)-(5). To optimize the weights it suffices to solve the system of recurrences obtained when putting into recurrence (5) all possible values of $d_F(v) \geq 2$, $d_A(u) \geq 2$ for every $u \in N_F(v)$ and $d_F(w)$ such that $1 \leq d_F(w) \leq d_F(v)$. The number of recurrences is infinite; fortunately one can restrict to the recurrences with $2 \leq d_F(v) \leq 4$, $2 \leq d_A(u) \leq 7$ for every $u \in N_F(v)$, and $1 \leq d_F(w) \leq d_F(v)$. Indeed, since $\Delta f_{d_A(v)} = 0$ for each free vertex u with $d_A(u) \geq 7$, all recurrences with $d_A(u) > 7$ for some vertex $u \in N_F(v)$ will be majorized by some recurrence with $d_A(u) = 7$. (See also the appendix for more details.) Similarly considering recurrence (2), under the assumption $a_1 \geq 0.75$ and $f_{\geq 5} = 1$, all recurrences with $d_A(u) \geq 7$ will be majorized by the one with $d_A(u) = 6$.

Thus the system of recurrences to be solved is finite. We use a program to generate those linear recurrences and try to remove superfluous ones. For more details about the choice of the recurrences we refer to the appendix.

For any fixed and valid choice of the 8 weights, this system of 210 recurrences is easy to solve. Finally using a program based on random local search, values of the 8 weights are searched as to minimize the bound on the running time. We numerically obtained the following values of the weights: $a_1 = 0.7655$, $a_2 = 0.9595$, $a_{\geq 3} = 1$, $f_2 = 0.3813$, $f_3 = 0.7485$, $f_4 = 0.9259$, $f_5 = 0.9880$, $f_{\geq 6} = 1$. Using these values an upper bound of $O(1.3387^n)$ is established.

There are two tight recurrences (i.e. ones for which the solution corresponds to the stated upper bound). One of these recurrences is obtained for the following configuration. There exists an available vertex v with $N_F(v) = \{u_i : 1 \leq i \leq 3\}$ of maximum F-degree. For all i,

$1 \leq i \leq 3$, $|N_A(u_i)| = 6$ and for any i_1, i_2 such that $1 \leq i_1, i_2 \leq 3$, $N_A(u_{i_1}) \cap N_A(u_{i_2}) = \{v\}$. Moreover for any available neighbour v' of a vertex u_i we have $d_F(v') = 3$. If in this configuration the algorithm branches on v using (B2), the corresponding recurrence is $P[\mu] \leq 1 + P[\mu - (a_3 + 3 \cdot f_{\geq 6} + 15 \cdot (a_{\geq 3} - a_2)] + P[\mu - (a_3 + 3 \cdot (f_{\geq 6} - f_5)]$, and this is a tight recurrence. The other tight recurrence is obtained when the algorithm `mdc` uses (B1) to branch on the available neighbours of a free vertex u with $d_A(u) = 4$. Then the corresponding recurrence is $P[\mu] \leq 1 + 4 \cdot P[\mu - 4 \cdot a_1 - f_4 - a_1]$.

Theorem 1. *Algorithm* `mdc` *solves problem* `MinDC` *in time* $O(1.3387^n)$.

It is not unlikely that the worst case running time of the Branch & Reduce algorithm `mdc` is $O(\alpha^n)$ for some $\alpha < 1.3387$. Significant improvements of the upper bound require a more clever choice of the measure or new techniques to analyse the running time of Branch & Reduce algorithms.

4 An Exponential Lower Bound

Since our upper bound on the running time of `mdc` might be significantly larger than its real worst case running time, it is natural to ask for a lower bound that may give an idea of how far the established upper bound of $O(1.3387^n)$ is from the real worst case running time of `mdc`.

Theorem 2. *The worst case running time of algorithm* `mdc` *is* $\Omega(1.2599^n)$.

Proof. To prove the claimed lower bound we consider the graphs G_k for integers $k \geq 1$ (see also Fig. 1).

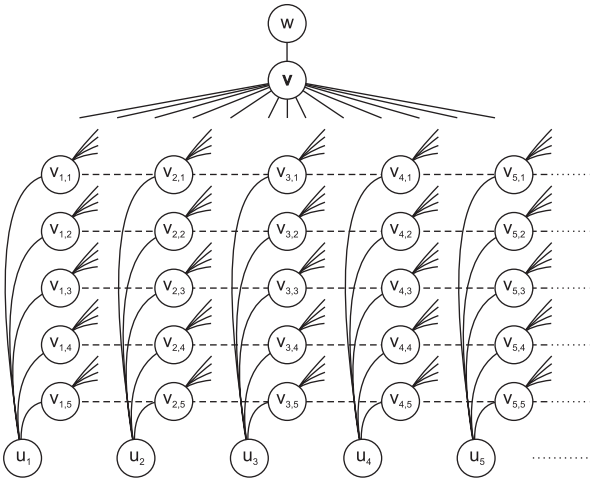

Figure 1: Graph G_k. (Dashed lines represent non edges.)

The vertex set of G_k is $V_k = \{w, v\} \cup \bigcup_{\substack{1 \leq i \leq k \\ 1 \leq j \leq 5}} \{v_{i,j}\} \cup \bigcup_{1 \leq i \leq k} \{u_i\}$. The edge set E_k of the graph G_k consists of the edge $\{w, v\}$ and the union of

$\bigcup_{\substack{1\le i\le k\\ 1\le j\le 5}} \big\{\{v, v_{i,j}\}, \{u_i, v_{i,j}\}\big\}$ and $\bigcup_{\substack{1\le i<i'\le k\\ 1\le j,j'\le 5}} \big\{\{v_{i,j}, v_{i',j'}\} : (i', j') \neq (i+1, j)\big\}$. Notice that the graph G_k has precisely $6k + 2$ vertices.

To establish an exponential lower bound of the worst case running time, we lower bound the number of leaves of a search tree obtained by an execution of `mdc` on G_k. (Notice that ties will be broken such as to maximise the number of leaves.) A vertex of minimum degree in G_k is w, and thus for our analysis we consider $\texttt{mdc}(G_k, S_1, D_1, A_1, F_1)$ with $S_1 = \{v\}$, $D_1 = \emptyset$, $A_1 = N_{G_k}(v)$ and $F_1 = V_k \setminus N_{G_k}[v] = \{u_1, u_2, \ldots, u_k\}$.

Notice that no reduction rules can be applied. For each i, every vertex $v_{i,j}$, $1 \le j \le 5$, has the unique non neighbor $v_{i+1,j}$ in $\{v_{i+1,j} : 1 \le j \le 5\}$. Furthermore since all available vertices have only one free neighbor, branching rule (B1) will be applied. We assume that whenever `mdc` branches on a subproblem using (B1), then it branches on the neighborhood of the free vertex u_i with the smallest possible value of i.

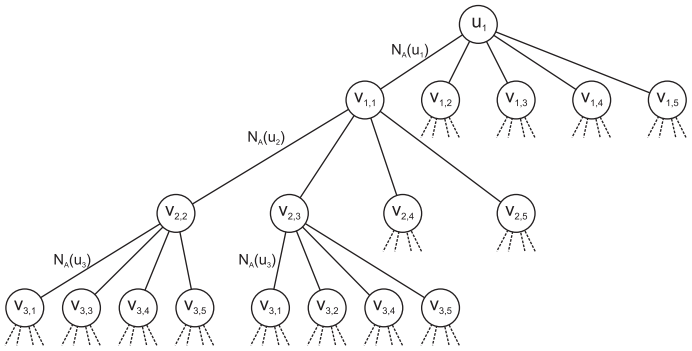

Figure 2: Part of the search tree.

Let us consider the first branching. By our rule, the algorithm branches on the neighborhood of u_1, i.e. $v_{1,j}$, $1 \le j \le 5$, into 5 subproblems. Consider a subproblem, say v_{1,j_1} had been selected, thus the partition of the subproblem is (S_2, D_2, A_2, F_2) with $S_2 = S_1 \cup \{v_{1,j_1}\}$, $D_2 = D_1 \cup \{v_{2,j_1}\} \cup \bigcup_{1\le j\le 5}\{v_{1,j} : j \neq j_1\}$, $A_2 = A_1 \setminus (\{v_{2,j_1}\} \cup \bigcup_{1\le j\le 5}\{v_{1,j}\})$ and $F_2 = F_1 \setminus \{u_1\}$.

All five subproblems have the same sets F_2 and almost the same A_2 and thus the graph remaining after removal of discarded vertices is essentially the same. By our construction the previous arguments apply to any partition (S_2, D_2, A_2, F_2), and thus `mdc` branches using (B1) on all *four* available neighbors of u_2. (Note that v_{2,j_1} is not adjacent to v_{1,j_1} and thus not available.)

Suppose now that for some l, $2 \le l \le k - 1$, (S_l, D_l, A_l, F_l) is the partition obtained from successive branchings on the available neighbors of $u_1, u_2, \ldots, u_{l-1}$ when applying the branching rule (B1). Moreover suppose that $A_l = \bigcup_{\substack{l\le i\le k\\ 1\le j\le 5}} \{v_{i,j}\} \setminus \{v_{l,m}\}$ for a certain $m \in \{1, 2, 3, 4, 5\}$ and $F_l = \bigcup_{l\le i\le k}\{u_i\}$. Then `mdc` applies neither halting nor reduction rules. In-

deed it is clear that (H1), (H2), (R1) and (R2) cannot be applied since each available vertex has precisely one free neighbor, and each free vertex has at least four available neighbours. The reduction rule (R3) cannot be applied since for any two available vertices u and v, there exist two vertices u' and v' in $G[A_l]$ such that $u' \notin N(u)$, $u \in N(v)$ and $v' \notin N(v)$, $v \in N(u)$ (e.g. for $v_{i,j}$ and $v_{i',j'}$ with $l \leq i, i' \leq k-1$, $1 \leq j, j' \leq 5$, $v_{i,j} \neq v_{i',j'}$, consider the vertices $v_{i+1,j}$ and $v_{i'+1,j'}$). The reduction rule (R4) is not applicable since each free vertex has its own available neighbors. Concerning (R5), for each free vertex u_i, $l \leq i \leq k-1$, each available neighbor $v_{i,j}$, $1 \leq j \leq 5$, is non adjacent to $v_{i+1,j}$ and $v_{i+1,j} \notin N_A(u_i)$. It follows that (R5) is not applicable. Thus `mdc` will apply (B1) on the neighbourhood of the vertex $u_i \in A_l$ for some i, and without loss of generality we suppose that it chooses the vertex $u_i \in A_l$ with i smallest as possible, i.e. $i = l$.

Then when branching on an available neighbor of u_l for a subproblem of the abovementioned type, `mdc` obtains a partition $(S_{l+1}, D_{l+1}, A_{l+1}, F_{l+1})$ with $A_{l+1} = \bigcup_{\substack{l+1 \leq i \leq k \\ 1 \leq j \leq 5}} \{v_{i,j}\} \setminus \{v_{l+1,m}\}$ for some $m \in \{1,2,3,4,5\}$ and $F_{l+1} = \bigcup_{l+1 \leq i \leq k} \{u_i\}$. This shows that `mdc` branches successively on the neighborhoods of the vertices $u_1, u_2, u_3, \ldots u_{k-1}$ using rule (B1).

Consequently, each application of rule (B1) branches into 4 subproblems (actually 5 for u_1). Thus the number of leaves in the search tree is $\Omega(4^{n/6}) = \Omega(1.2599^n)$, where $n = 6k+2$ is the number of vertices of G_k. □

5 An Exponential Space Algorithm

Robson has shown in [22], how the worst case running time of a polynomial space Branch & Reduce algorithm can be reduced at the cost of using exponential space. This technique is called *memorization* (see also [5]). The main principle is to store the solutions of all the solved subproblems recursively in an exponential-size database. Then when solving a new subproblem, if the algorithm needs to compute the solution of a subproblem encountered before, the already computed solution is obtained from the database in polynomial time (see [22]).

We apply memorization to our polynomial space algorithm `mdc` and call the new algorithm `mdc-exp`.

Theorem 3. *The algorithm* `mdc-exp` *solves the problem* `MinDC` *in time* $O(1.3234^n)$ *using exponential space.*

Proof. Each time `mdc-exp` solves a subproblem (G, S, D, A, F), we store the corresponding solution in a database. That is, we store the size of a minimum clique being a subset of A which dominates all vertices in F. (One might also store a minimum clique itself.) Note that for a fixed set $S \cup D$ the subsets $A \subseteq V$ and $F \subseteq V$ are uniquely determined. In fact, given a graph $G = (V, E)$ and a vertex $v \in S$ (as e.g. the first vertex added to S when

`mdc` is called), when considering the set $U = A \cup F$, it is easy to see that $A = N(v) \cap U$ and $F = U \setminus N(v)$. Consequently, for any graph $G = (V, E)$ and any vertex $v \in V$, there are at most 2^n subsets of vertices $U \subseteq V$, and thus the database contains no more than 2^n entries.

Similar to the analysis of `mdc` in section 3, we denote by $P[n]$ the maximum number of subproblems recursively solved by `mdc-exp` to compute a solution on an instance of size n. Let $P_i[n]$, $i \leq n$, be the maximum number of subproblems (G, S, D, A, F) solved when `mdc-exp` runs on a graph with n vertices and $i = |W|$ where

$$W = \{u \in A \ : \ d_F(u) \geq 1\} \cup \{u \in F \ : \ d_A(u) \geq 2\}.$$

(Note that any vertex in $(A \cup F) \setminus W$ is handled by a reduction rule of `mdc`, and thus any instance having such a vertex can be reduced, possibly by a sequence of reductions, to a smaller instance with $(A \cup F) \setminus W = \emptyset$.) According to the measure μ used in the running time analysis of `mdc`, a_1 and f_2 are the smallest weights in μ, and hence the removal of k vertices from W decreases the measure by at least $(k \cdot \min\{a_1, f_2\})$. Thus it follows that $P_i[n] \leq \alpha^{n-i\min\{a_1,f_2\}}$, where $O(\alpha^n)$ is an upper bound on the running time of `mdc` established by considering the measure μ.

Moreover, there are $\binom{n}{i}$ subsets of vertices of cardinality i, and so $P_i[n] \leq \binom{n}{i}$, for all $i \leq n$, since each problem is solved at most once. Consequently $P_i[n] \leq \min\{\alpha^{n-i\min\{a_1,f_2\}}, \binom{n}{i}\}$. To find the largest value of $P_i[n]$ for an $i \in [0, n]$, we need to know the value(s) of β satifying $\alpha^{n-\beta n\min\{a_1,f_2\}} = \binom{n}{\beta n}$ where $i = \beta n$, $0 \leq \beta \leq 1$. (See Figure 3.)

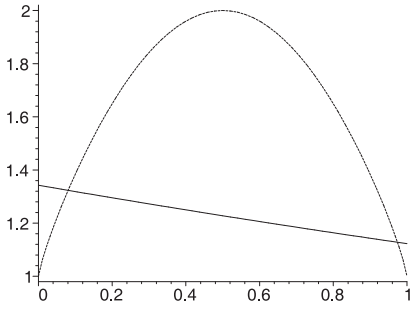

Figure 3: Plain and dotted curves represent the functions $f(\beta) = 1.342521^{1-0.6068\beta}$ and $g(\beta) = 1/(\beta^\beta(1-\beta)^{1-\beta})$ for $0 \leq \beta \leq 1$. Since f is a strictly decreasing function on $[0, 1]$, we are interested in the smallest value of $\beta \in [0, 1]$ such that $f(\beta) = g(\beta)$.

Hence we have $(\alpha^{1-\beta\min\{a_1,f_2\}})^n = (1/(\beta^\beta(1-\beta)^{1-\beta}))^n$ which implies $\alpha^{1-\beta\min\{a_1,f_2\}} = 1/(\beta^\beta(1-\beta)^{1-\beta})$. Then, by summing over all possible values of i we obtain that $P[n] = \sum_{i=0}^{n} P_i[n] = O(n \cdot P_{\beta n}[n])$.

Using the values $a_1 = 0.7457$, $a_2 = 0.9580$, $a_{\geq 3} = 1$, $f_2 = 0.6068$, $f_3 = 0.8675$, $f_4 = 0.9780$, $f_5 = 0.9973$, $f_{\geq 6} = 1$ for the weights in μ, we obtain an upper bound of $O(\alpha^n)$ for `mdc` where $\alpha = 1.342521$.

And finally one obtains that $0.08057 > \beta > 0.08056$ where β satisfies $1.342521^{1-0.6068\beta} = 1/(\beta^\beta(1-\beta)^{1-\beta})$. This implies the claimed bound on the running time of `mdc-exp`. □

6 The Algorithm of Culberson et al.

In [2] Culberson et al. propose the Branch & Reduce algorithm `DomClq` to decide whether a given graph has a dominating clique or not. The algorithm starts by branching on all neighbors of a vertex w of minimum degree, i.e. for each neighbor v of w the algorithm calls `DomClq`$(G, \{v\}, N(v), V \setminus N[v])$.

Algorithm DomClq(G, S, A, F)
Input: **A graph** $G = (V, E)$**,** $S \subseteq V$ **set of selected vertices,** $A \subseteq V$ **set of available vertices,** $F \subseteq V$ **set of free vertices.**
Output: **A boolean indicating whether** G **has a dominating clique or not.**

```
if F = ∅ then
    DOMCLQ = S
    return yes
Find w ∈ F such that |N_A(w)| = min_{v∈F} |N_A(v)|
P = N_A(w)
if P = ∅ then return no
A'' = A
while P ≠ ∅ do
    Select v ∈ P
    S = S ∪ {v}
    A' = N_{A''}(v)
    F' = F \ N_F[v]
    if DomClq(G, S, A', F') then return yes
    S = S \ {v}
    A'' = A'' \ {v}
    P = P \ {v}
return no
```

The main interest of their work is in determining the phase transition of the existence problem `ExDC` on random graphs in the classical $G_{n,p}$ model. They establish a threshold of $\frac{3-\sqrt{5}}{2}$. For experimental studies they use algorithm `DomClq`. Culberson et al. report on experiments for random graphs with up to 1000 vertices (and p close to the threshold $\frac{3-\sqrt{5}}{2}$). A theoretical analysis of the worst case running time had not been attempted.

It is natural to compare the algorithm `DomClq` and the algorithm `mdc`. However as Fomin et al. point out in [4], upper bounds of Branch & Reduce algorithms are likely to overestimate the real (and unknown) worst case running time. Thus comparing upper bounds of the worst case running time of both algorithms might lead to wrong conclusions. We provide a lower bound of the worst case running time of the algorithm of Culberson et al. that will be of great help when comparing the two algorithms.

Theorem 4. *The worst case running time of algorithm* `DomClq` *is* $\Omega(1.4142^n)$.

Proof. The graphs used to demonstrate the lower bound are denoted by $G_{k,\ell} = (V_{k,\ell}, E_{k,\ell})$, $k, \ell \geq 1$. Their vertex set is $V_{k,\ell} = \{w, v, a, b\} \cup \{x_i : 0 \leq i \leq \ell\} \cup \{y_i : 1 \leq i \leq \ell\} \cup \{u_i : 1 \leq i \leq k\} \cup \{s_i : 1 \leq i \leq k+\ell-1\}$. Thus $G_{k,\ell}$ has $n = 2k+3\ell+4$ vertices. The edge set of $G_{k,\ell}$ is $E_{k,\ell} = \{\{w, v\}\} \cup \{\{v, x_i\} : 0 \leq i \leq \ell\} \cup \{\{v, y_i\} : 1 \leq i \leq \ell\} \cup \{\{v, s_i\} : 1 \leq i \leq k+\ell-1\} \cup \bigcup_{i=1}^{k} \{\{u_i, s_j\} : i \leq j \leq i + \ell - 1\} \cup \{\{a, x_i\} : 0 \leq i \leq \ell\} \cup \{\{b, y_i\} : 1 \leq i \leq \ell\}$. Finally all pairs of vertices in the set $S = \{s_i : 1 \leq i \leq k + \ell - 1\} \cup \{x_i : 0 \leq i \leq \ell\} \cup \{y_i : 1 \leq i \leq \ell\}$ are adjacent in $G_{k,\ell}$ with the exception of the following pairs $\{\{s_i, x_0\} : i \mod \ell \neq 0\} \cup \{\{x_i, y_j\} : 1 \leq i, j \leq \ell\}$. See also the graph $G_{k,5}$ in Fig. 4.

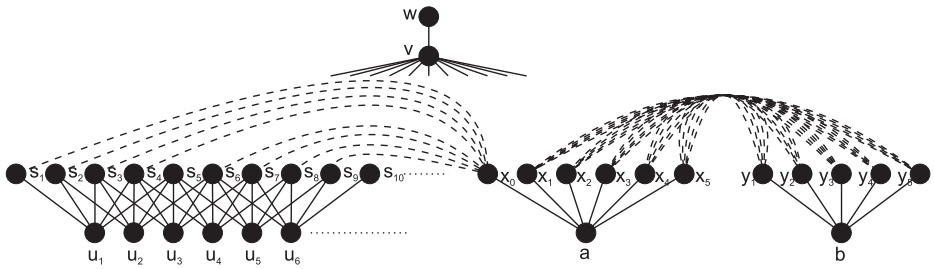

Figure 4: Graph $G_{k,5}$. (Dashed lines represent non edges of $G_{k,5}$.)

Let C be a dominating clique of $G_{k,\ell}$. Clearly, C contains the vertex b or one of its neighbors. However, $b \in C$ is impossible since a could not have a neighbor in C in this case. By the construction of $G_{k,\ell}$, all vertices y_i have the same neighborhood, thus without lost of generality we may assume $y_1 \in C$. This implies $x_0 \in C$ since x_0 is the only common neighbor of a and y_1. Consequently, and that is important for the analysis, the vertex x_0 must belong to every dominating clique, and therefore no vertex s_i with $i \mod \ell \neq 0$ belongs to a dominating clique of $G_{k,\ell}$.

Consider the algorithm `DomClq` on the previous page. When applied to a graph $G_{k,\ell}$, first a vertex of minimum degree is determined and this is w and then the algorithm branches on v which is the unique neighbor of w in $G_{k,\ell}$. Thus `DomClq`$(G_{k,\ell}, \{v\}, S, \{u_i : 1 \leq i \leq k\} \cup \{a, b\})$ is called. Hence `DomClq` chooses a vertex in $F = \{u_i : 1 \leq i \leq k\} \cup \{a, b\}$ with smallest number of neighbors in S. Suppose that each time the algorithm has to do such a choice it chooses the free vertex u_i with the smallest possible index i. Then the algorithm branches on the neighbors of u_i. Note that, according to the above remark, no vertex s_i with $i \mod \ell \neq 0$ belongs to a dominating clique, and thus `DomClq` really branches on each of the five neighbors of u_i.

Consider now the search tree obtained after branching on the neighbors of $u_1, u_2, \ldots, u_{t-1}$, $2 \leq t \leq k$. When branching on the neighbors of u_t the set of vertices already discarded from the original graph $G_{k,\ell}$ is $\{w, v\} \cup \{u_j : 1 \leq j < t\} \cup \{s_j : 1 \leq j < t\}$. How many subproblems are obtained by branching on the neighbors of u_t?

- when branching on s_t (the first neighbor of u_t), the algorithm discards 2 vertices (since $N[s_t] = \{s_t, u_t\}$), and thus obtains a subproblem of size $q-2$,
- when branching on s_{t+1} (the second neighbor of u_t), the algorithm discards 4 vertices: $s_t, s_{t+1}, u_t, u_{t+1}$, and thus obtains a subproblem of size $q-4$,
- when branching on s_{t+2} (the third neighbor of u_t), the algorithm discards 6 vertices: $s_t, s_{t+1}, s_{t+2}, u_t, u_{t+1}, u_{t+2}$, and thus obtains a subproblem of size $q-6$,
- ...
- when branching on $s_{t+\ell-1}$ (the last neighbor of u_t), the algorithm removes 2ℓ vertices: $s_t, s_{t+1}, s_{t+2}, \ldots, s_{t+\ell-1}, u_t, u_{t+1}, u_{t+2}, \ldots, u_{t+\ell-1}$, and thus obtains a subproblem of size $q-2\ell$,

We denote by $L[q]$ the maximum number of leaves of the search tree when applying `DomClq` to an induced subgraph of $G_{k,\ell}$ on $q \leq n$ vertices. According to our previous analysis, we obtain the following recurrence

$$L[q] \geq L[q-2] + L[q-4] + L[q-6] + \cdots + L[q-2\ell].$$

Solving this recurrence by substituting $L[q] = \alpha^q$ one obtains $L[q] \geq 1.4142^q$.

Consequently, 1.4142^n is a lower bound for the maximum number of leaves of the search tree of an execution of algorithm `DomClq` on an input graph with n vertices. □

Thus the lower bound of the worst case running time of algorithm `DomClq` is asymptotically larger than the upper bound of the running time of algorithm `mdc`. Consequently our algorithm has a better worst case running time.

7 Conclusions

There are many interesting questions related to exact exponential time algorithms for dominating clique problems. The $O(1.3387^n)$ time polynomial space algorithm `mdc` and the $O(1.3234^n)$ time exponential space algorithm `mdc-exp` solve the problem `MinDC` and can also be applied to solve `ExDC`. It would be interesting to establish algorithms for `ExDC` being asymptotically faster than our algorithms. Currently the best known running time of an exact algorithm for `MaxDC` is $3^{n/3}\, n^{O(1)}$. We conjecture that there is a faster algorithm for that problem.

The problems `MaxDC` and `ExDC` are polynomial time solvable on chordal graphs by enumerating all the at most n maximal cliques and verifying which

of them are dominating sets. On the other hand, `MinDC` is NP-hard on split graphs. Using the Minimum Set Cover algorithms given in [4] one easily obtains an $O(1.2303^n)$ algorithm for `MinDC` on split graphs. This approach can be extended to chordal graphs without increasing the running time. What about exact algorithms for `MinDC` and `ExDC` on weakly triangulated graphs?

Clearly all the three considered dominating clique problems are easier on sparse graphs, i.e. polynomial or subexponential time solvable depending on the applied definition of sparseness. It would be interesting to study those problems on dense graphs.

Acknowledgment. We are grateful to the anonymous referees for their helpful comments.

References

[1] Cozzens, M. B., and L. L. Kelleher, Dominating cliques in graphs, *Discrete Mathematics* **86** (1990), pp. 101–116.

[2] Culberson, J., Y. Gao, and C. Anton, Phase Transitions of Dominating Clique Problem and Their Implications to Satisfiability Search, *Proceedings of IJCAI 2005*, pp. 78–83.

[3] Downey, R. G., and M. R. Fellows, *Parameterized complexity*, Springer-Verlag, New York, 1999.

[4] Fomin, F. V., F. Grandoni, and D. Kratsch, Measure and conquer: Domination - A case study, *Proceedings of ICALP 2005*, Springer-Verlag, 2005, Berlin, LNCS 3380, pp. 192–203.

[5] Fomin, F. V., F. Grandoni, and D. Kratsch, Some new techniques in design and analysis of exact (exponential) algorithms, *Bulletin of the EATCS* **87** (2005), pp. 47–77.

[6] Fomin, F. V., F. Grandoni, and D. Kratsch, Solving Connected Dominating Set Faster than 2^n, *Proceedings of FSTTCS 2006*, Springer-Verlag, 2006, Berlin, LNCS 4337, pp. 152–163.

[7] Fomin, F. V., D. Kratsch, and G. J. Woeginger, Exact (exponential) algorithms for the dominating set problem, *Proceedings of WG 2004*, Springer-Verlag, 2004, Berlin, LNCS 3353, pp. 245–256.

[8] Garey, M. R. and D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*. W.H. Freeman and Co., San Francisco, 1979.

[9] Gaspers, S., D. Kratsch, and M. Liedloff, Exponential time algorithms for the minimum dominating set problem on some graph classes, *Proceedings of SWAT 2006*, Springer-Verlag, 2006, Berlin, LNCS 4059, pp. 148–159.

[10] Gaspers, S., and M. Liedloff, A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs, *Proceedings of WG 2006*, Springer-Verlag, 2006, Berlin, LNCS 4271, pp. 78–89.

[11] Grandoni, F., A note on the complexity of minimum dominating set, *Journal of Discrete Algorithms* **4** (2006), pp. 209–214.

[12] Haynes, T. W., S. T. Hedetniemi, and P. J. Slater, *Fundamentals of domination in graphs*. Marcel Dekker Inc., New York, 1998.

[13] Haynes, T. W., S. T. Hedetniemi, and P. J. Slater, *Domination in graphs: Advanced Topics*. Marcel Dekker Inc., New York, 1998.

[14] Held, M. and R.M. Karp, A dynamic programming approach to sequencing problems, *Journal of SIAM*, pp. 196–210, 1962.

[15] Iwama, K., Worst-case upper bounds for ksat, *Bulletin of the EATCS* **82** (2004), pp. 61–71.

[16] Johnson, D. S., M. Yannakakis, and C. H. Papadimitriou, On generating all maximal independent sets, *Information Processing Letters* **27** (1988), pp. 119–123.

[17] Kratsch, D., Algorithms, in *Domination in Graphs: Advanced Topics*, T. Haynes, S. Hedetniemi, P. Slater, (eds.), Marcel Dekker, 1998, pp. 191–231.

[18] Lawler, E. L., A note on the complexity of the chromatic number problem, *Information Processing Letters* **5** (1976), pp. 66–67.

[19] Moon, J. W., and L. Moser, On cliques in graphs, *Israel Journal of Mathematics* **3** (1965), pp. 23–28.

[20] Randerath, B., and I. Schiermeyer, Exact algorithms for Minimum Dominating Set, Technical Report zaik-469, Zentrum für Angewandte Informatik, Köln, Germany, April 2004.

[21] Razgon, I., Exact computation of maximum induced forest, *Proceedings of SWAT 2006*, Springer-Verlag, 2006, Berlin, LNCS 4059, pp. 160–171.

[22] Robson, J. M., Algorithms for maximum independent sets, *Journal of Algorithms* **7** (1986), pp. 425–440.

[23] Schöning, U., Algorithmics in exponential time, *Proceedings of STACS 2005*, Springer-Verlag, 2005, Berlin, LNCS 3404, pp. 36–43.

[24] Tarjan, R. and A. Trojanowski, Finding a maximum independent set, *SIAM Journal on Computing* **6** (1977), pp. 537–546.

[25] Villanger, Y., Improved exponential-time algorithms for treewidth and minimum fill-in. *Proceedings of LATIN 2006* Springer-Verlag, 2005, Berlin, LNCS 3887, pp. 800–811.

[26] West, D., *Introduction to Graph Theory*, Prentice Hall, 1996.

[27] Woeginger, G. J., Exact algorithms for NP-hard problems: A survey, *Combinatorial Optimization - Eureka, You Shrink!*, Springer-Verlag, 2003, Berlin, LNCS 2570, pp. 185–207.

[28] Woeginger, G. J., Space and time complexity of exact algorithms: Some open problems, *Proceedings of IWPEC 2004*, Springer-Verlag, 2004, Berlin, LNCS 3162, pp. 281–290.

Appendix: Simplifying the system of recurrences and lower bounding Δ_{IN} and Δ_{OUT}

To compute (optimal) values for the weights used in the definition of the non standard measure, one has to generate a system of recurrences and solve it. We have seen that the number of those recurrences is infinite; and that, fortunately, by restricting the ranges of some degrees and by putting some restrictions on the weights, it is possible to reduce the recurrences to be considered such that only a finite (still a possibly large) number remains. In this appendix we explain how to further reduce the number of recurrences to facilitate the computation of optimal weights.

We recall the recurrence corresponding to the branching rule (B2):

$$P[\mu] \leq 1 + P[\mu - \Delta_{OUT}] + P[\mu - \Delta_{IN}]. \tag{5}$$

where

$$\Delta_{IN} \geq a_{d_F(v)} + \sum_{u \in N_F(v)} f_{d_A(u)} + \sum_{w \in N_A(N_F(v)) \setminus \{v\}} (a_{d_F(w)} - a_{d_F(w)-|N_F(v) \cap N_F(w)|}) \tag{3}$$

$$\Delta_{OUT} \geq a_{d_F(v)} + \sum_{u \in N_F(v) \setminus N_{F2}(v)} \Delta f_{d_A(u)} + \sum_{u \in N_{F2}(v)} f_{d_A(u)} + \sum_{w \in N_A(N_{F2}(v)) \setminus \{v\}} a_{d_F(w)} \tag{4}$$

The above lower bound of Δ_{IN} seems to require that all possible subgraphs of G induced by $\{v\} \cup N_F(v) \cup N_A(N_F(v))$ are to be considered when generating the recurrences. Even when taking into account only those edges of G with one end point in $N_F(v)$ and one in $N_A(N_F(v))$; this would still generate a huge number of recurrences. Therefore we shall establish a lower bound for Δ_{IN} which rescues us from considering the vertices of $N_A(N_F(v))$. Using it the number of recurrences reduces drastically.

Let $G = (V, E)$ be a graph, (S, D, A, F) a partition of its vertex set, and $\mu(G) = \mu(G, S, D, A, F)$ the measure of G under partition (S, D, A, F) as defined in Section 3. Let d be the maximum F-degree of an available vertex of G. In the remainder we consider the changes when branching rule (B2) is applied to G, and show how to obtain lower bounds for Δ_{IN}, and Δ_{OUT}.

Lower bounding Δ_{IN}.

When applying (B2) and selecting a vertex v of maximum F-degree, the gain in the measure is obtained by the removal of v and $N_F(v)$, and by the decrease of the F-degrees of the vertices in $N_A(N_F(v)) \setminus \{v\}$. To establish a lower bound for Δ_{IN}, we shall lower bound the sum of $a_{d_F(w)} - a_{d_F(w)-|N_F(v) \cap N_F(w)|}$ taken over all $w \in N_A(N_F(v)) \setminus \{v\}$.

Let $w \in A$ be a vertex whose h free neighbours, $h \leq d_F(w) \leq d$, are removed. We denote by $W_{h,d}$ our lower bound of the gain in Δ_{IN} obtained by the decrease of the F-degree of w, or the removal of w by a reduction rule applied to the subproblem (branch IN). Using the lower bounds $W_{h,d}$ the following lower bound of Δ_{IN} is established.

$$\Delta_{IN} \geq a_{d_F(v)} + \sum_{u \in N_F(v)} f_{d_A(u)} + \sum_{w \in N_A(N_F(v)) \setminus \{v\}} W_{|N_F(w) \cap N_F(v)|, d_F(v)}$$

The following table lists the values of $W_{h,d}$ depending on h and d.

(h,d)	$W_{h,d}$
$(1,1)$	a_1
$(1,2)$	$\min(a_1, a_2 - a_1)$
$(1,3)$	$\min(a_1, a_2 - a_1, a_{\geq 3} - a_2)$
$(2,2)$	a_2
$(2,3)$	$\min(a_2, a_{\geq 3} - a_1)$
$(2,4)$	$\min(a_2, a_{\geq 3} - a_1, a_{\geq 3} - a_2)$
$(3,3)$	$a_{\geq 3}$
$(3,4)$	$\min(a_{\geq 3}, a_{\geq 3} - a_1)$
$(3,5)$	$\min(a_{\geq 3}, a_{\geq 3} - a_1, a_{\geq 3} - a_2)$
otherwise	0

Correctness of the stated values is easy to prove. Clearly, zero is a trivial lower bound of $W_{h,d}$ for all (h,d). If $(h,d) = (1,1)$ then w has only one neighbour in F and we remove this neighbour; consequently w would be removed from G by applying the reduction rule (R1) and we gain at least a_1. If $(h,d) = (1,2)$ then w has one free neighbour (see case $(h,d) = (1,1)$) or w has two free neighbours whose one is removed and the free degree of w decreases from a_2 to a_1. If $(h,d) = (2,3)$ then either $d_F(w) = 2$, we remove this two neighbours and by applying (R1) w is removed; or $d_F(w) = 3$, then removing two of the neighbours would lead to $a_{\geq 3} - a_1$. The others cases can be proved in a similar way.

To improve further on the lower bound, we impose some restricitions on the weights still to be computed:

$$\forall k \;\; 2 \leq k \leq d \quad \Rightarrow \quad 2W_{1,k} \leq W_{2,k}$$

$$\forall k \;\; 3 \leq k \leq d \quad \Rightarrow \quad 3W_{1,k} \leq W_{3,k}$$

These conditions permit to generate a smaller number of recurrences than those obtained by the direct enumeration of all recurrences. In fact, those conditions guarantee that the gain $W_{h,d}$ is smaller when, for two (respectively three) disctinct available vertices exactly one free neighbour is removed, than when exactly two (respectively three) free neighbours of one available vertex are removed.

For $i \geq 0$, let λ_i be the number of those available vertices $w \in N_A(N_F(v)) \setminus \{v\}$ for which precisely i free neighbours are removed from the graph when rule (B2) is applied to G and vertex $v \in A$ is selected. Due to the above conditions, for any positive integer d, and all $\lambda_i \geq 0$ we have :

$$\begin{aligned}
\sum_{i\geq 0} \lambda_i W_{i,d} &\geq \sum_{1\leq i\leq 3} \lambda_i W_{i,d} \\
&= \lambda_1 W_{1,d} + \lambda_2 W_{2,d} + \lambda_3 W_{3,d} \\
&= \lambda_1 W_{1,d} + 2\lambda_2/2 W_{2,d} + 3\lambda_3/3 W_{3,d} \\
&\geq \lambda_1 W_{1,d} + 2\lambda_2 W_{1,d} + 3\lambda_3 W_{1,d} \\
&= W_{1,d} \sum_{i=1}^{3} i\lambda_i
\end{aligned}$$

Consequently, each $w \in N_A(N_F(v)) \setminus \{v\}$ contributes 1 for each vertex in $N_F(v)$ to the sum $\sum_{i=1}^{3} i\lambda_i$, and thus we obtain the same value when each vertex $u \in N_F(v)$ contributes 1 for each w in $N_A(N_F(v)) \setminus \{v\}$. Note that, if $d_F(v) \leq 3$ then $\sum_{i=1}^{3} i\lambda_i = \sum_{u\in N_F(v)}(d_A(u) - 1)$; and if $d_F(v) > 3$ then $W_{1,d_F(v)} = 0$.

$$\Delta_{IN} \geq a_{d_F(v)} + \sum_{u\in N_F(v)} f_{d_A(u)} + W_{1,d_F(v)} \sum_{u\in N_F(v)} (d_A(u) - 1) \qquad (6)$$

Finally we obtained a lower bound of Δ_{IN} that needs to consider only v, its free neighbours, and their A-degrees. As the computation shows this lower bound is good enough to establish a good upper bound on the running time of our algorithm, when combining it with the lower bound of Δ_{OUT} obtained in the next subsection.

Lower bounding Δ_{OUT}.

The basic idea is the same as in the previous subsection. We want to establish a lower bound of Δ_{OUT} that uses only the vertex v, its free neighbours, and their A-degrees.

The vertex $v \in A$ is discarded when applying rule (B2) and one obtains subproblem (branch OUT). Recall that we denoted by $N_{F2}(v)$ the set $\{u \in N_F(v) : d_A(u) = 2\}$, and let $d_{F2}(v)$ be the cardinality of $N_{F2}(v)$.

When removing vertex $v \in A$, each vertex $u \in N_F(v)$ having A-degree two remains with only only available neighbour. Thus reduction rule (R2) will select the unique A-neighbour and discard u. Let $Q = N_A(N_{F2}(v)) \setminus \{v\}$ be the set of vertices removed by applying (R2) to subproblem (branch OUT). By $W_{d_{F2}(v),d}$ we denote our lower bound on the gain obtained by removing all the vertices of Q.

We distinguish three cases. Throughout we use the notation $\lambda_i = |\{v \in Q \ : \ d_F(v) = i\}|$, $i \geq 0$.

$d = 2$ Clearly the gain obtained by removing the vertices of Q is $\lambda_1 a_1 + \lambda_2 a_2$.

$$\begin{aligned} \lambda_1 a_1 + \lambda_2 a_2 &= \lambda_1 a_1 + 2\lambda_2 a_2/2 \\ &\geq \lambda_1 \min(a_1, a_2/2) + 2\lambda_2 \min(a_1, a_2/2) \\ &\geq d_{F2}(v) \min(a_1, a_2/2) \end{aligned}$$

Thus $W_{d_{F2}(v),2} = d_{F2}(v) \min(a_1, a_2/2)$.

$d = 3$ The gain obtained by the removal of the vertices in Q is $\lambda_1 a_1 + \lambda_2 a_2 + \lambda_3 a_{\geq 3}$, and we obtain

$$\begin{aligned} \lambda_1 a_1 + \lambda_2 a_2 + \lambda_3 a_{\geq 3} &= \lambda_1 a_1 + 2\lambda_2 a_2/2 + 3\lambda_3 a_{\geq 3}/3 \\ &\geq \lambda_1 \min(a_1, a_2/2, a_{\geq 3}/3) + 2\lambda_2 \min(a_1, a_2/2, \\ &\quad a_{\geq 3}/3) + 3\lambda_3 \min(a_1, a_2/2, a_{\geq 3}/3) \\ &\geq d_{F2}(v) \min(a_1, a_2/2, a_{\geq 3}/3) \end{aligned}$$

Thus $W_{d_{F2}(v),3} = d_{F2}(v) \min(a_1, a_2/2, a_{\geq 3}/3)$.

$d > 3$ The gain by the removal of the vertices in Q is at least $W_{d_{F2}(v),d} = \min(d_{F2}(v) \min(a_1, a_2/2, a3/3), a_{\geq 3})$ for all $d > 3$. To see this notice that if $|Q| = 1$ all vertices in $N_{F2}(v)$ are adjacent to the unique vertex in Q having a large F-degree.

Remark. *Since $a_1 \leq a_2 \leq a_{\geq 3}$, if $d_{F2}(v) = 1$ the gain is at least a_1 and if $d_{F2}(v) = 2$ the gain is at least $\min(2a_1, a_2)$, i.e. for all $d, W_{1,d} = a_1$ and for all $d, W_{2,d} = \min(2a_1, a_2)$*

Consequently we obtain the following lower bound of Δ_{OUT}.

$$\Delta_{OUT} \geq a_{d_F(v)} + \sum_{u \in N_F(v) \setminus N_{F2}(v)} \Delta f_{d_A(u)} + \sum_{u \in N_{F2}(v)} f_{d_A(u)} + W_{d_{F2}(v), d_F(v)} \quad (7)$$

The lower bounds in this appendix have been used to generate a system of recurrences, which in turn has been solved to obtain the weights and the running time mentioned in Section 3.