REPORTS IN INFORMATICS

ISSN 0333-3590

A Branch-and-Reduce Algorithm for Finding a Minimum Independent Dominating Set in Graphs

Serge Gaspers and Mathieu Liedloff

REPORT NO 344 **January 2007**

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL
http://www.ii.uib.no/publikasjoner/texrap/pdf/2007-344.pdf

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at
http://www.ii.uib.no/publikasjoner/texrap/.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

A Branch-and-Reduce Algorithm for Finding a Minimum Independent Dominating Set in Graphs

Serge Gaspers* Mathieu Liedloff†

Abstract

An independent dominating set $\mathcal{D}$ of a graph $G = (V, E)$ is a subset of vertices such that every vertex in $V \setminus \mathcal{D}$ has at least one neighbour in $\mathcal{D}$ and $\mathcal{D}$ is an independent set, i.e. no two vertices in $\mathcal{D}$ are adjacent. Finding a minimum independent dominating set in a graph is an NP-hard problem. Whereas it is hard to cope with this problem using parameterized and approximation algorithms, there is a simple exact $O(1.4423^n)$-time algorithm solving the problem by enumerating all maximal independent sets. In this paper we improve the latter result, providing the first non trivial algorithm computing a minimum independent dominating set of a graph in time $O(1.3575^n)$. Furthermore, we give a lower bound of $\Omega(1.3247^n)$ on the worst-case running time of this algorithm, showing that the running time analysis is almost tight. Finally we show that for the class of c-dense graphs (graphs respecting $|E| \geq c|V|^2$ for a constant c, $0 < c < 1/2$) an $O(1.3575^{n\sqrt{1-2c}})$-time algorithm solves the problem.

1 Introduction

During the last years the interest in the design of exact exponential time algorithms has been growing significantly. Nice surveys have been written on this subject. In one due to Woeginger [24], the author emphasizes the major techniques used to design exact exponential time algorithms. We also refer the reader to the recent survey of Fomin et al. [10] discussing some new techniques in the design of exponential time algorithms. In particular they discuss Measure & Conquer and lower bounds.

The problem MINIMUM INDEPENDENT DOMINATING SET (MIDS) is also known as MINIMUM MAXIMAL INDEPENDENT SET, since every independent dominating set is a maximal independent set. This problem asks for a set of minimum cardinality that is both independent and dominating. Whereas MAXIMUM INDEPENDENT SET and MINIMUM DOMINATING SET have been studied very deeply in the field of exact algorithms, the best known exact algorithm for MIDS trivially enumerates all maximal independent sets.

Known results. A set $\mathcal{I} \subseteq V$ of a graph $G = (V, E)$ is independent if no two vertices in $\mathcal{I}$ are adjacent. The problem of finding a Maximum Independent Set (MIS) of a graph was among the first problems shown to be NP-hard [12].

It is known that a MIS of a graph on n vertices can be computed in $O(1.4423^n)$ time by combining a result due to Moon and Moser, who showed in 1965 [18] that the number of maximal independent sets of a graph is upper bounded by $3^{n/3}$, and a result due to Johnson, Yannakakis and Papadimitriou, providing in [16] a polynomial delay algorithm to generate all maximal independent sets. Moreover many exact algorithms for this problem have been published, starting in 1977 by an $O(1.2600^n)$ algorithm by Tarjan and Trojanowski [23]. The best known algorithms for MIS until now are an $O(1.2108^n)$ algorithm by Robson [20] in

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway. serge.gaspers@ii.uib.no

†Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France. liedloff@univ-metz.fr

1986, a very long algorithm of running time $O(1.1889^n)$ by Robson [21] in 2001 and a very simple algorithm with running time $O(1.2210^n)$ by Fomin et al. [8] in 2006.

A set $\mathcal{D} \subseteq V$ of a graph $G = (V, E)$ is dominating if every vertex in $V \setminus \mathcal{D}$ has at least one neighbour in $\mathcal{D}$. The problem of finding a Minimum Dominating Set (MDS) of a graph is well known to be NP-hard [12].

Until recently, the only known exact exponential time algorithm to solve MDS asked for trivially enumerating the 2^n subsets of vertices. The year 2004 saw a particular interest in providing some faster algorithms for solving this problem. Indeed, three papers with exact algorithms for MDS were published. In [11] Fomin et al. present an $O(1.9379^n)$ time algorithm, in [19] Randerath and Schiermeyer establish an $O(1.8899^n)$ time algorithm and Grandoni [14] obtains an $O(1.8026^n)$ time algorithm.

By now, the fastest published algorithm is due to Fomin et al. [9]. They use the Measure & Conquer approach to obtain an algorithm with running time $O(1.5263^n)$ and using polynomial space. By applying a memorization technique they show that this running time can be reduced to $O(1.5137^n)$ when allowing exponential space usage.

A natural and well studied combination of these two problems asks for a subset of vertices of minimum cardinality that is both dominating and independent. This problem is called MINIMUM INDEPENDENT DOMINATING SET (MIDS).

It is known that a Minimum Independent Dominating Set ($MIDS$) can be found in polynomial time for several graph classes like interval graphs [3], chordal graphs [7], cocomparability graphs [17] and AT-free graphs [2], whereas the problem remains NP-complete for bipartite graphs [4] and comparability graphs [4]. Concerning inapproximability results, Halldórsson established in [15] that there is no constant $\epsilon > 0$ such that MIDS can be approximated within a factor of $n^{1-\epsilon}$ in polynomial time, assuming $P \neq NP$. The same inapproximation result even holds for circle graphs and bipartite graphs [5].

To the best of our knowledge, the only exact exponential time algorithm for MIDS has been observed by Randerath and Schiermeyer [19]. They use the result due to Moon and Moser [18] as explained previously and an algorithm enumerating all the maximal independent sets to obtain an $O(1.4423^n)$ time algorithm for MIDS.

Recently, the problem has also been considered in Parameterized Approximability. Downey et al. have shown in [6] that it is $W[2]$-hard to approximate k-INDEPENDENT DOMINATING SET within a factor $g(k)$, for any computable function $g(k) \geq k$. This means that, unless $W[2] = FPT$, there is no algorithm with running time $O(f(k) \cdot n^{O(1)})$ (where $f(k)$ is any computable function independent of n) which either asserts that there is no independent dominating set of size at most k for a given graph G, or otherwise asserts that there is one of size at most $g(k)$, for any computable function $g(k) \geq k$.

Our results. In this paper we present an $O(1.3575^n)$ time algorithm for solving MIDS using the Measure & Conquer approach to analyze its running time. As the bottleneck of the algorithm in [19] are the vertices of degree two, we develop several methods to handle them more efficiently such as marking some vertices and a sophisticated reduction rule described in section 3.1. Combined with some elaborated branching rules, this enables us to lower bound shrewdly the progress made by the algorithm at each branching step, and thus to obtain an algorithm which improves the best known result from $O(1.4423^n)$ to $O(1.3575^n)$. Furthermore, we obtain a very close lower bound of $\Omega(1.3247^n)$ on the running time of our algorithm, which is very rare for non trivial exponential time algorithms. We also show that the algorithm can be improved when the input graph has many edges and give an algorithm in time $O(1.3575^{n\sqrt{1-2c}})$ for MIDS on c-dense graphs.

This paper is organized as follows. In section 2, we introduce the necessary concepts and definitions. Section 3 presents the algorithm for MIDS on general graphs. We prove its correctness and an upper bound on its worst-case running time in section 4. In section 5, we establish a lower bound on its worst-case running time, which is very close to the upper bound. An algorithm for MIDS on c-dense graphs is given in section 6 and we conclude with

section 7.

2 Preliminaries

Let $G = (V, E)$ be an undirected and simple graph. For a vertex $v \in V$ we denote by $N(v)$ the neighbourhood of v and by $N[v] = N(v) \cup \{v\}$ the closed neighbourhood of v. The degree $d(v)$ of v is the cardinality of $N(v)$. For a given subset of vertices $S \subseteq V$, $G[S]$ denotes the subgraph of G induced by S, $N(S)$ denotes the set of neighbours in $V \setminus S$ of vertices in S and $N[S] = N(S) \cup S$. We also define $N_S(v)$ as $N(v) \cap S$ and $d_S(v)$ (called the S-degree of v) as the cardinality of $N_S(v)$. In the same way, given two subsets of vertices $S \subseteq V$ and $X \subseteq V$, we define $N_S(X) = N(X) \cap S$.

A *clique* is a set $S \subseteq V$ of pairwise adjacent vertices. A graph $G = (V, E)$ is called *bipartite* if V admits a partition into two independent sets. A bipartite graph $G = (V, E)$ is a *complete bipartite graph* if every vertex of one independent set is adjacent to every vertex of the other independent set. A *connected component* of a graph is a maximal subset of vertices inducing a connected subgraph.

In a branch-and-reduce algorithm the current problem is divided into smaller ones such that an optimal solution, if one exists, occurs in at least one subproblem. If the algorithm considers only one subproblem in a given case, we refer to a reduction rule, otherwise to a branching rule.

Consider a vertex $u \in V$ of degree two with two non adjacent neighbours v_1 and v_2. In such a case, a branch-and-reduce algorithm will typically branch into three subcases when considering u: either u or v_1 or v_2 are in the solution set. In the third branch, one can consider that v_1 is not in the solution set as this is already considered by the second branch. In order to memorize that v_1 is not in the solution set but still needs to be dominated, we mark v_1.

Definition 1. A *marked graph* $G = (F, M, E)$ is a triple where $F \cup M$ denotes the set of vertices of G and E denotes the set of edges of G. The vertices in F are called *free vertices* and the ones in M *marked vertices*.

Definition 2. Given a marked graph $G = (F, M, E)$, an independent dominating set $\mathcal{D}$ of G is a subset of free vertices, i.e. $\mathcal{D} \subseteq F$, such that $\mathcal{D}$ is an independent dominating set of the graph $G' = (F \cup M, E)$.

Remark. It is possible that such an independent dominating set does not exist in a marked graph, namely if a marked vertex has no free neighbours.

Finally to close this section we introduce the notion of an *induced marked subgraph*.

Definition 3. Given a marked graph $G = (F, M, E)$ and two subsets $S, T \subseteq (F \cup M)$, an *induced marked subgraph* $G[S, T]$ is the marked graph $G' = (S, T, E')$ where $E' \subseteq E$ are the edges of G with both end points in $S \cup T$.

Note that notions like neighbourhood and degree in a marked graph $G = (F, M, E)$ are the same as in the corresponding simple graph $G = (F \cup M, E)$.

3 Computing a $MIDS$ on Marked Graphs

In this section we present an algorithm solving `MIDS` on marked graphs.

From the previous definitions it follows that a subset $\mathcal{D} \subseteq V$ is a $MIDS$ of a graph $G' = (V, E)$ if and only if $\mathcal{D}$ is a $MIDS$ of the marked graph $G = (V, \emptyset, E)$. Hence the algorithm of this section is able to solve the problem on simple graphs as well.

Given a marked graph $G = (F, M, E)$, consider the graph $G[F]$ induced by its free vertices. In the following subsection we introduce a reduction rule which deletes a connected component of $G[F]$ which is a clique.

3.1 Eliminating Cliques in $G[F]$

Consider the function **RedClique**. Given a marked graph $G = (F, M, E)$ and a clique $C \subseteq F$, this function removes $N[C]$ from G and adds some marked vertices such that a $MIDS$ of this new graph union one vertex from C equals a $MIDS$ of G.

Function RedClique($G = (F, M, E), C \subseteq F$)
Input: A marked graph $G = (F, M, E)$ **and a clique** $C \subseteq F$ **such that** C **is a connected component of** $G[F]$**.**
Output: A marked graph $G' = (F', M', E')$ **s.t.** G' **has the properties defined in Lemma 4.**

```
if |C| = 1 then
    G' ← G[F \ C, M \ N(C)];
else
    if ∃v ∈ C s.t. N_M(v) = ∅ then
        G' ← RedClique(G[F − {v}, M], C − {v})
    else
        let N(C) = {h_1, h_2, ..., h_k}
        H ← ∅
        for i ← 1 to k − 1 do
            for j ← i + 1 to k do
                if N_C(h_i) ∩ N_C(h_j) = ∅ then
                    add to H a new marked vertex h_{i,j}
        G' = (F', M', E') ← G[F \ C, M \ N(C)]
        M' ← M' ∪ H
        foreach h_{i,j} ∈ H do
            foreach v ∈ N_F(N[C]) s.t. {v, h_i} ∈ E or {v, h_j} ∈ E do
                E' ← E' ∪ {v, h_{i,j}}
return G'
```

Lemma 4. *Let $G = (F, M, E)$ be a marked graph and C a connected component of $G[F]$ which is a clique. The function* **RedClique** *computes in polynomial time a marked graph $G' = (F', M', E')$ such that:*

(i) the size of a $MIDS$ of G is equal to the size of a $MIDS$ of G' plus one, if G admits an independent dominating set,

(ii) $F' = F \setminus C$,

(iii) no edge of $E' - E$ has both end points in F', i.e. the function adds no edge between two free vertices.

Proof. First, note that whenever there is a clique component C in $G[F]$, every independent dominating set contains exactly one vertex of C. Indeed, at least one vertex of C has to be taken in the independent dominating set to dominate C and at most one vertex in C can be taken because the solution has to be an independent set.

If $|C| = 1$, the unique vertex in C must be part of the $MIDS$. So, the function just deletes C and its neighbourhood (since these vertices are dominated). By now we assume that $|C| \geq 2$.

If there is a vertex $v \in C$ with no marked neighbour, then we will not choose this vertex in the $MIDS$. As a matter of fact, every vertex in C dominates C. So, a vertex in C which also dominates some marked vertices is always a better choice than a vertex that does not. Consequently, the function just deletes v and calls itself recursively on the clique component $C - \{v\}$.

Assume now that $|C| \geq 2$ and that every vertex in C has at least one neighbour in M. Then, the function will create one new marked vertex $h_{i,j}$ for every two vertices $h_i, h_j \in N(C)$ that do not share a same neighbour in C. It replaces $N[C]$ by these new marked vertices. A vertex $h_{i,j}$ will be adjacent to a vertex $v \in F \setminus C$ iff h_i or h_j was adjacent to v. So, when all vertices $h_{i,j}$ will be dominated by vertices in $F \setminus C$ in G', at least all the vertices in $N(C)$ except the neighbours of one vertex $u \in C$ are dominated in G. It is then clear among which vertices of C to choose the vertex to include in the $MIDS$. And whenever a vertex $h_{i,j}$ is not dominated in G', no vertex of C can dominate all undominated vertices in $N(C)$ in G.

Remark that, once all these new marked vertices are dominated, it is possible to determine in polynomial time which vertex of the clique C must be added to the solution in order to obtain a $MIDS$ for the initial marked graph. Note that there can be several equivalent choices.

As $N[C]$ is deleted from the original graph, we have $F' = F - C$. The function does not create new edges between two free vertices because the only new edges created during the computation join free and new marked vertices. It is not hard to see that **RedClique** has polynomial running time. □

3.2 The Algorithm

In this subsection, we give the algorithm **ids** computing the size of a $MIDS$ of a marked graph (see next page). The branching rules are quite complicated but it is fairly simple to check that the algorithm computes the size of a $MIDS$ (if one exists). It is not difficult to transform **ids** into an algorithm that actually outputs a $MIDS$. In the next section we prove the correctness and give a detailed analysis of **ids**.

4 Correctness and Analysis of the Algorithm

Intuitively, marked vertices do not make the instance of the problem more difficult: they cannot be taken in the $MIDS$ and the only thing they are good for is to put restrictions on their free neighbours. Moreover, free vertices having only marked neighbours can be handled without branching. So, it is an advantage when the F-degree of a vertex decreases. We will therefore assign different weights to the free vertices according to their F-degree.

Let n_i denote the number of free vertices having F-degree i. For the running time analysis we consider the following measure of the size of G:

$$k = k(G) = \sum_{i \geq 0} w_i n_i \leq n$$

where the weights $w_i \in [0, 1]$. In order to simplify the running time analysis, we make the following assumptions:

- $w_0 = 0$,
- $w_i = 1$ for $i \geq 3$,
- $w_1 \leq w_2$,
- $\Delta w_1 \geq \Delta w_2 \geq \Delta w_3$ where $\Delta w_i = w_i - w_{i-1}, i \in \{1, 2, 3\}$.

Theorem 5. *Algorithm* **ids** *solves the minimum independent dominating set problem in time* $O(1.3575^n)$.

Algorithm ids(G)
Input: A marked graph $G = (F, M, E)$.
Output: The size of a $MIDS$ **of** G.

if $F = M = \emptyset$ **then**
  return 0 **(0)**
else if $\exists u \in M$ *s.t.* $d_F(u) = 0$ **then**
  return ∞ **(1)**
else if $\exists u \in M$ *s.t.* $d_F(u) = 1$ **then**
  let v **be the unique free neighbour of** u
  return $1 + \mathbf{ids}(G[F \setminus N[v], M \setminus N(v)])$ **(2)**
else if $\exists C \subseteq F$ *s.t.* C *is a clique* $\wedge N_F(C) = \emptyset$ **then**
  return $1 + \mathbf{ids}(\mathbf{RedClique}(G, C))$ **(3)**
else if $\exists B \subseteq F$ *s.t.* B *induces a complete bipartite graph* $\wedge N_F(B) = \emptyset$ **then**
  let B **be partitioned into two independent sets** X **and** Y
  return min{ $|X| + \mathbf{ids}(G[F \setminus N[X], M \setminus N(X)])$; **(4)**
   $|Y| + \mathbf{ids}(G[F \setminus N[Y], M \setminus N(Y)])$}
else if $\exists C \subseteq F$ *s.t.* C *is a clique* $\wedge |C| \geq 3 \wedge \exists! v \in C$ *s.t.* $d_F(v) \geq |C|$ **then**
  return min{ $1 + \mathbf{ids}(G[F \setminus N[v], M \setminus N(v)])$; **(5)**
   $\mathbf{ids}(G[F \setminus \{v\}, M \cup \{v\}, E])$}
else
  choose $u \in F$ **of minimum** F**-degree with a neighbour in** F **of maximum** F**-degree**
  if $d_F(u) = 1$ **then**
   return $1 + $ **min**{ $\mathbf{ids}(G[F \setminus N[u], M \setminus N(u)])$; **(6)**
    $\mathbf{ids}(G[F \setminus N[N_F(u)], M \setminus N(N_F(u))])$}
  else if $d_F(u) = 2$ **then**
   let $N_F(u) = \{v_1, v_2\}$
   return $1 + $ **min**{ $\mathbf{ids}(G[F \setminus N[u], M \setminus N(u)])$; **(7)**
    $\mathbf{ids}(G[F \setminus N[v_1], M \setminus N(v_1)])$;
    $\mathbf{ids}(G[F \setminus (N[v_2] \cup \{v_1\}), (M \cup \{v_1\}) \setminus N(v_2)]$}
  else
   choose $v \in F$ **of maximum** F**-degree**
   return min{ $1 + \mathbf{ids}(G[F \setminus N[v], M \setminus N(v)])$; **(8)**
    $\mathbf{ids}(G[F \setminus \{v\}, M \cup \{v\}])$}

Proof. Let $P[k]$ denote the number of subproblems recursively solved to compute a solution for an instance of size k. As the time spent in each call of **ids**, excluding the time spent by the corresponding recursive calls, is polynomial, it is sufficient to show that for a valid choice of the weights, $P[k] = O(1.3575^n)$.
We will analyse the nine cases of algorithm **ids** one by one. Cases (0) to (3) are reduction rules and the other cases correspond to branching rules.

case (0) If the set of vertices is empty, the algorithm returns 0 since no vertex can be added to the independent dominating set any more.

case (1) If there is a marked vertex u having no free neighbour, u has no possibility to be dominated and thus the algorithm returns ∞, meaning that there is no solution for this subproblem.

case (2) If there is a marked vertex u with only one free neighbour v, the only possibility for u to be dominated is to add v to the $MIDS$. Consequently, $N[v]$ is deleted from the graph.

case (3) If there is a clique C of free vertices which are not adjacent to any other free vertices, we use the function of Lemma 4 to remove C. Since the number of free vertices decreases by $|C|$ and no new edges are added between any two free vertices, the F-degrees of

the remaining free vertices do not increase. Thus the measure k does not increase. (Note that the number of marked vertices and their F-degree can increase by this reduction, but these parameters do not occur in our measure.)

Note that, in cases (2) and (3), the number of free vertices strictly decreases. This means that the number of consecutive applications of these reduction rules to a subproblem is at most n. Moreover, the measure of the problem instance does not increase in these cases. Thus, $P[k]$ can at most increase by a linear factor due to these reduction rules and cases (2) and (3) do not contribute to the exponential factor in $P[k]$.

case (4) If there is a subset B of free vertices such that $G[B]$ induces a complete bipartite graph and no vertex of B is adjacent to a free vertex outside B, then the algorithm branches into two subcases. Let X and Y be the two maximal independent sets of $G[B]$. Then a $MIDS$ contains either X or Y. In both cases we delete B and the marked neighbours of either X or Y. The smallest possible subset B satisfying the conditions of this case is a P_3, i.e. a path of three vertices, as any smaller complete bipartite component in F is handled by case (3). Since we only count the number of free vertices, we obtain the following recurrence:

$$P[k] \leq 2P[k - 2w_1 - w_2]. \tag{1}$$

This means that the algorithm solves two subproblems in this case and in each of them, at least two vertices of degree at least one and one vertex of degree at least 2 are removed. It is clear that any complete bipartite component with more than three vertices would lead to a better recurrence.

case (5) If there is a subset C of at least three free vertices which form a clique and only one vertex $v \in C$ has free neighbours outside C, the algorithm either includes v in the solution set or it excludes this vertex by marking it. In the first case, all the neighbours of v are deleted (including C). In the second case, v is marked and the $C - \{v\}$ clique component appears in $G[F]$. Then $C - \{v\}$ will be deleted by the reduction rule of case (3). In both cases, C is deleted and in the first case, the neighbours of v outside C are also deleted (at least one free vertex of F-degree at least one). So we have:

$$P[k] \leq P[k - w_1 - 2w_2 - w_3] + P[k - 2w_2 - w_3]. \tag{2}$$

case (6) If there is a free vertex u such that $d_F(u) = 1$, a $MIDS$ either includes u or its free neighbour v in the solution. Vertex v cannot have F-degree one because this would have been handled by case (3). For the analysis, we consider two cases:

1. $d_F(v) = 2$. Let x denote the other free neighbour of v. Note that $d_F(x) \neq 1$ as this would have been handled by case (4). We consider again two subcases:

   (a) $d_F(x) = 2$. When u is chosen in the independent dominating set, u and v are deleted and the degree of x decreases to one. When v is chosen in the independent dominating set, u, v and x are deleted from the marked graph. So, we obtain the following recurrence for this case:

   $$P[k] \leq P[k - 2w_2] + P[k - w_1 - 2w_2]. \tag{3}$$

   (b) $d_F(x) \geq 3$. Vertices u and v are deleted in the first branch, and u, v and x are deleted in the second branch. The recurrence for this subcase is:

   $$P[k] \leq P[k - w_1 - w_2] + P[k - w_1 - w_2 - w_3]. \tag{4}$$

2. $d_F(v) \geq 3$. At least one free neighbour of v has F-degree at least 2. Otherwise case (4) would have been applied. Therefore the recurrence for this subcase is:

   $$P[k] \leq P[k - w_1 - w_3] + P[k - 2w_1 - w_2 - w_3]. \tag{5}$$

case (7) If there is a free vertex u such that $d_F(u) = 2$ and none of the above cases apply, the algorithm branches into three subcases. Let v_1 and v_2 be the two free neighbours of u. Either u belongs to the $MIDS$, or v_1 is taken in the $MIDS$, or v_1 is being marked and v_2 is taken in the $MIDS$. We distinguish two cases:

1. $d_F(v_1) = d_F(v_2) = 2$. In this case, due to the choice of the vertex u by the algorithm, all free vertices of this connected component T in $G[F]$ have F-degree 2. T cannot be a C_4, i.e. a cycle of 4 vertices, as this is a complete bipartite graph and would have been handled by case (4).

   (a) Suppose that T is a C_5. Let the vertices of T be ordered (u, v_1, x_1, x_2, v_2). When u is taken in the $MIDS$, u, v_1, v_2 are deleted and in the next recursive call, case (3) is applied for the clique $\{x_1, x_2\}$ and thus, x_1 and x_2 will also be deleted. When v_1 is taken in the $MIDS$, three vertices are again deleted and case (3) will be applied for $\{v_2, x_2\}$. When v_2 is taken in the $MIDS$, $N[v_2]$ is deleted and v_1 becomes marked. In the next recursive call, x_1 will be taken in the $MIDS$ by case (2). In every recursive call, T is entirely deleted:

   $$P[k] \leq 3P[k - 5w_2]. \tag{6}$$

   (b) Suppose that T is a C_6. Let the vertices of T be ordered $(u, v_1, x_1, y, x_2, v_2)$. When u is taken in the $MIDS$ u, v_1, v_2 are deleted and in the next recursive call, case (4) will be applied for $\{x_1, y, x_2\}$ and thus, the algorithm will branch into two subcases, both deleting x_1, y and x_2. When v_1 is taken in the $MIDS$, three vertices are again deleted and case (4) will be applied for $\{v_2, x_2, y\}$. When v_2 is taken in the $MIDS$, $N[v_2]$ is deleted and v_1 becomes marked. In the next recursive call, x_1 will be taken in the $MIDS$ by case (2). Finally in each of the 5 recursive calls, T is entirely deleted, thus:

   $$P[k] \leq 5P[k - 6w_2]. \tag{7}$$

   (c) Suppose that T is a C_7. Let the vertices of T be labeled $(u, v_1, x_1, y_1, y_2, x_2, v_2)$ in clockwise order. When u is chosen in the $MIDS$ u, v_1, v_2 are deleted and the F-degrees of x_1, x_2 decrease by one. We obtain a similar situation when branching on v_1: three vertices are deleted and the F-degrees of two vertices decrease to one. When the algorithm chooses v_2 in the $MIDS$, v_1 is marked and x_1 must be added to the $MIDS$ by case (2) and y_2 will then be added by case (3). Consequently, the algorithm deletes the C_7 entirely and we obtain the recurrence:

   $$P[k] \leq 2P[k + 2w_1 - 5w_2] + P[k - 7w_2]. \tag{8}$$

   (d) Suppose now that T is a C_l, $l \geq 8$. Using the same arguments as in the previous cases, it is not hard to check that we obtain the following recurrence:

   $$P[k] \leq 2P[k + 2w_1 - 5w_2] + P[k + 2w_1 - 8w_2]. \tag{9}$$

2. Without loss of generality, suppose now that $d_F(v_1) \geq 3$. We analyze two subcases:

   (a) $d_F(v_2) = 2$. In this subcase, v_1 and v_2 are not adjacent, otherwise case (5) could have been applied. Let x_3 denote the other neighbour of v_2. Recall that due to the choice of u by the algorithm $\forall y \in F$, $d_F(y) \geq d_F(u)$. If $d_F(x_3) = 2$, as previously we branch on u, v_1 and v_2, and we get the following recurrence:

   $$P[k] \leq P[k+w_1-3w_2-w_3]+P[k+w_1-4w_2-w_3]+P[k-3w_2-w_3]. \tag{10}$$

And if $d_F(x_3) \geq 3$, let q denote the number of vertices in $N_F(v_1)$ with F-degree at least 3. In the worst case $q < 3$ and branching on u, v_1 and v_2, we obtain the following recurrence for $q \in \{0, 1, 2\}$:

$$\begin{aligned} P[k] \quad \leq \quad & P[k + (2-q)w_1 - (4-q)w_2 - w_3] + \\ & P[k + w_1 - (4-q)w_2 - (1+q)w_3] + P[k - 2w_2 - 2w_3]. \end{aligned} \tag{11}$$

(b) $d_F(v_2) \geq 3$. If v_1 and v_2 are not adjacent, branching on u, v_1 and v_2 leads to the following recurrence:

$$P[k] \leq P[k - w_2 - 2w_3] + P[k - 3w_2 - w_3] + P[k - 3w_2 - 2w_3]. \tag{12}$$

However if v_1 and v_2 are adjacent, let $x_1 \in N_F(v_1) \setminus \{u, v_2\}$. We consider two possible cases:

i. if $d_F(x_1) = 2$, we obtain:

$$P[k] \leq P[k + w_1 - 2w_2 - 2w_3] + 2P[k - 2w_2 - 2w_3]. \tag{13}$$

ii. if $d_F(x_1) \geq 3$. Let $x_2 \in N_F(v_2) \setminus \{u, v_1\}$. If $d_F(x_2) = 2$, then:

$$\begin{aligned} P[k] \quad \leq \quad & P[k + w_1 - 2w_2 - 2w_3] + P[k + w_1 - 2w_2 - 3w_3] + \\ & P[k - 2w_2 - 2w_3]. \end{aligned} \tag{14}$$

However if $d_F(x_2) \geq 3$ we get the following recurrence:

$$P[k] \leq P[k - w_2 - 2w_3] + 2P[k - w_2 - 3w_3]. \tag{15}$$

case (8) In this case the algorithm either takes v in the $MIDS$ or marks it, i.e. v does not belong to the $MIDS$. We consider two cases:

1. $d_F(v) = 3$. In this case, regarding the previous rules handled by the algorithm, every free vertex has degree three. $N_F[v]$ cannot be a clique, otherwise case (3) would have been applied. So, at least two vertices in $N_F(v)$ have a neighbour outside $N_F[v]$ (remark that this could be the same vertex). This implies that if the algorithm takes v in the $MIDS$, the F-degree of at least two free vertices decreases to two in the worst case (if $|N_F(N_F[v])| = 1$ then the decrease of the measure would be higher since $\Delta w_2 + \Delta w_3 \geq 2\Delta w_3$ because of the conditions on the weights). If the algorithm marks v, then three free vertices get F-degree two. The recurrence for this case is:

$$P[k] \leq P[k + 2w_2 - 6w_3] + P[k + 3w_2 - 4w_3]. \tag{16}$$

2. $d_F(v) \geq 4$. When v is taken in the $MIDS$, at least five free vertices are deleted. When v is marked, the measure decreases by w_3. Thus we have this recurrence:

$$P[k] \leq P[k - 5w_3] + P[k - w_3]. \tag{17}$$

Finally the values of weights are computed by a random local search for minimizing the bound on the running time. Using the values $w_1 = 0.8588$ and $w_2 = 0.9630$ for the weights, one can easily verify that $P[k] = O(1.3575^n)$. □

The tight recurrences of the latter proof (i.e. the worst case recurrences) (15) and (16) correspond to cases where there are many vertices of F-degree 3 in the local structure the algorithm considers.

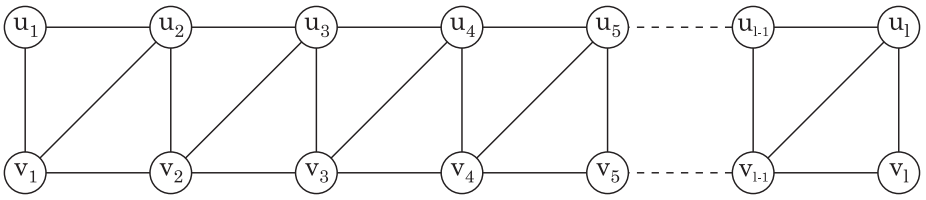

Figure 1: graph G_l

5 A Lower Bound on the Running Time of the Algorithm

In order to analyze the progress of the algorithm during the computation of a $MIDS$, we used a non standard measure. In this way we have been able to determine an upper bound on the size of the subproblems recursively solved by the algorithm, and consequently we obtained an upper bound on the worst case running time. However the use of another measure could provide a "better upper bound" without changing the algorithm but only improving the analysis.

How far is the given upper bound of Theorem 5 from the best upper bound we can hope to obtain? In this section, we establish a lower bound on the worst case running time of our algorithm. This lower bound gives a really good estimation on the precision of the analysis. For example, in [9] Fomin et al. obtain a $O(1.5263^n)$ time algorithm for solving the dominating set problem and they exhibit a construction of a family of graphs giving a lower bound of $\Omega(1.2599^n)$ for its running time. They say that the upper bound of many exponential time algorithms is likely to be overestimated only due to the choice of the measure for the analysis of the running time, and they note the gap between their upper and lower bound for their algorithm. However, for our algorithm we have the following result:

Theorem 6. *Algorithm* **ids** *solves the minimum independent dominating set problem in time* $\Omega(1.3247^n)$.

To prove Theorem 6 on the lower bound of the worst-case running time of algorithm **ids**, consider the graph $G_l = (V_l, E_l)$ (see Fig. 1) defined by:

- $V_l = \{u_i, v_i : 1 \leq i \leq l\}$,
- $E_l = \{u_1, v_1\} \cup \big\{\{u_i, v_i\}, \{u_i, u_{i-1}\}, \{v_i, v_{i-1}\}, \{u_i, v_{i-1}\} : 2 \leq i \leq l\big\}$.

We denote by $G'_l = (V, \emptyset, E)$ the marked graph corresponding to the graph $G_l = (V, E)$.

For a marked graph $G = (F, M, E)$ we define $\delta_F = \min_{u \in F} |d_F(u)|$ and $MinU = \{u \in F \text{ s.t. } d_F(u) = \delta_F\}$ as the set of free vertices with smallest F-degree.

We denote the highest F-degree of the free neighbours of the vertices in $MinU$ by $\Delta MaxV = \max\big\{|d_F(v)| : v \in N_F(MinU)\big\}$.

Let $CandidateCase7 = \{u \in MinU : \exists v \in N_F(u) \text{ s.t. } d_F(v) = \Delta MaxV\}$ be the set of candidate vertices that **ids** can choose in case (7). W.l.o.g. suppose that when $|CandidateCase7| \geq 2$ and **ids** would apply case (7), it chooses the vertex with smallest index (e.g. if $CandidateCase7 = \{u_1, v_l\}$, the algorithm would choose u_1).

Lemma 7. *Let G'_l be the input of algorithm* **ids***. Suppose that* **ids** *only applies case (7) in each recursive call (with respect to the previous rule for choosing a vertex). Then, at each call of* **ids** *where the remaining input graph has more than four vertices, one of the following two properties is fulfilled:*

(1) $CandidateCase7 = \{u_k, v_l\}$ for a certain k, $1 \leq k \leq l-1$, and

(i) the set of vertices $\bigcup_{1 \leq i < k} \{u_i, v_i\}$ has been deleted from the input graph, and

(ii) all vertices in $\bigcup_{k\leq i\leq l}\{u_i, v_i\}$ *remain free in the input graph.*

(2) $CandidateCase7 = \{v_k, v_l\}$ *for a certain* k, $1 \leq k \leq l-1$, *and*

(i) the set of vertices $\{u_k\} \cup \bigcup_{1\leq i<k}\{u_i, v_i\}$ *has been deleted from the input graph, and*

(ii) all vertices in $\{v_k\} \cup \bigcup_{k<i\leq l}\{u_i, v_i\}$ *remain free in the input graph.*

Proof. We prove this result by induction. It is not hard to see that $CandidateCase7 = \{u_1, v_l\}$ for G'_l and that property (1) is verified.

Suppose now that property (1) is fulfilled. Then there exists an integer k, $1 \leq k \leq l-1$, such that $CandidateCase7 = \{u_k, v_l\}$. Since **ids** applies case (7) respecting the rule for choosing the vertex in $CandidateCase7$, the algorithm chooses vertex u_k. Then we branch into three sub-problems:

(b1) take u_k in the $MIDS$ and remove $N[u_k]$, thus the remaining free vertices are $\{v_{k+1}\} \cup \bigcup_{k+1<i\leq l}\{u_i, v_i\}$ whereas all other vertices are removed. Moreover for this remaining sub-problem, we obtain $CandidateCase7 = \{v_{k+1}, v_l\}$. So property (2) is verified. (Note also that $|N[u_k] \cap \bigcup_{k\leq i\leq l}\{u_i, v_i\}| = 3$.)

(b2) take v_k in the $MIDS$ and remove $N[v_k]$: $\bigcup_{k+2\leq i\leq l}\{u_i, v_i\}$ is the set of the remaining free vertices and all other vertices are removed. For the remaining sub-problem we obtain $CandidateCase7 = \{u_{k+2}, v_l\}$ and property (1) is verified. (Note also that $|N[v_k] \cap \bigcup_{k\leq i\leq l}\{u_i, v_i\}| = 4$.)

(b3) take u_{k+1} in the $MIDS$ and remove $N[u_{k+1}]$: the remaining free vertices are $\{v_{k+2}\} \cup \bigcup_{k+2<i\leq l}\{u_i, v_i\}$ and all other vertices are removed. For this remaining sub-problem we obtain $CandidateCase7 = \{v_{k+2}, v_l\}$ and property (2) is verified. (Note also that $|N[u_{k+1}] \cap \bigcup_{k\leq i\leq l}\{u_i, v_i\}| = 5$.)

If we suppose now that property (2) is fulfilled, branching on a vertex v_k gives us the same kind of subproblems. □

We prove now that computing a $MIDS$ of the graph G_l using algorithm **ids** involves to apply case (7) as long as the remaining graph has "enough" vertices.

Lemma 8. *Given the graph* G'_l *as input, as long as the remaining graph has more than four vertices, algorithm* **ids** *applies case (7) in each recursive call.*

Proof. We prove this result also by induction. First, when the input of the algorithm is the graph G'_l, it is clear that neither of cases (1) to (6) can be applied. So, case (7) is applied since $CandidateCase7 \neq \emptyset$ according to Lemma 7.

Consider now a graph obtained from G'_l by repeatingly branching using case (7). By Lemma 7, the remaining graph has no marked vertices (this excludes that case (1) and (2) are applied). It has no clique component induced by the set of free vertices since the graph is connected and there is no edge between u_{l-1} and v_l (this exclude case (3)). The free vertices do not induce a bipartite graph since $\{v_{l-1}, u_l, v_l\}$ induces a C_3 (this excludes case (4)). There is no clique C such that only one vertex of C has neighbours outside C: the largest induced clique in the remaining graph has size 3 and each of these cliques has at least two vertices having some neighbours outside the clique (this excludes case (5)). Also, according to Lemma 7, the remaining graph has no vertex of degree 1 (this excludes case (6)) and $CandidateCase7 \neq \emptyset$. Consequently, the algorithm applies case (7). □

Figure 2 gives a part of the search tree illustrating the fact that our algorithm recursively branches in three sub-problems with respect to case (7).

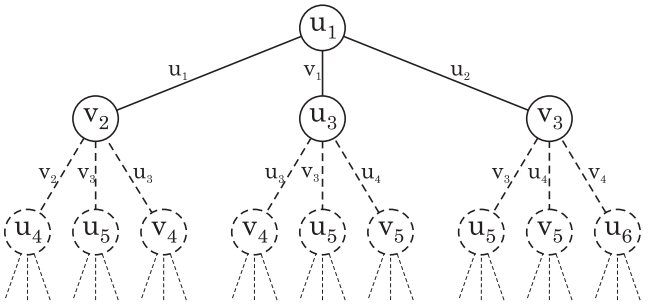

Figure 2: a part of the search tree

Proof of Theorem 6. Consider the graph G'_l and the search tree which results of branchings using case (7) until k vertices, $1 \leq k \leq 2l$, have been removed from the given input graph G'_l (G'_l has $2l$ vertices). Denote by $L[k]$ the number of leaves in this search tree. It is not hard to see that this leads to the following recurrence (see the notes in the proof of lemma 7):

$$L[k] = L[k-3] + L[k-4] + L[k-5]$$

and therefore $L[k] \geq 1.3247^k$. Consequently 1.3247^n is a lower bound of the maximum number of leaves that a search tree for **ids** could contain given an input graph having n vertices. □

6 An algorithm for c-dense graphs

Several problems are known to be NP-complete on graphs having a large number of edges [13, 22]. Some of them are Dominating Set, Independent Set, Hamiltonian Circuit and Hamiltonian Path. A convenient technique to prove that a problem is NP-hard on c-dense graphs, for a c with $0 < c < 1/2$, is to construct a graph G' by adding a (sufficiently) large component to G such that G' is c-dense.

Theorem 9. *For any constant c, $0 < c < 1/2$, the problem to decide whether a c-dense graph has an independent dominating set of size at most k is NP-complete.*

Proof. Let c be a constant such that $0 < c < 1/2$. It is well known that the decision problem Independent Dominating Set is in NP, and thus Independent Dominating Set on c-dense graphs is also in NP. We provide a polynomial many-one reduction from Independent Dominating Set to Independent Dominating Set on c-dense graphs. Let $G = (V, E)$ be a graph and k be an integer. We construct now a c-dense graph $G_c = (V_c, E_c)$ with $|E_c| \geq c \cdot |V_c|^2$. The graph G_c is obtained from G by adding a clique C of size $\lceil (1 + 4c|V| + \sqrt{1 + 8c|V|(1 + |V|) + 8|E|(2c - 1)})/(2 - 4c) \rceil$ to G. Note that the number of edges of G_c is greater than $c(|V| + |C|)^2$, and hence G_c is a c-dense graph. It remains to show that G has an independent dominating set of size at most k is and only if G_c has an independent dominating set of size at most $k + 1$.

First, assume that $\mathcal{D}$ is an independent dominating set of G of size at most k. Since the clique C in G_c has no neighbour in V, the vertices in C still need to be dominated. By adding only one vertex u of C to $\mathcal{D}$, the set $\mathcal{D} \cup \{u\}$ is an independent dominating set of G_c respecting $|\mathcal{D} \cup \{u\}| \leq k + 1$.

Conversely, suppose that $\mathcal{D}$ is an independent dominating set of $G_c = (V \cup C, E_c)$ of size at most k. Since C is a connected component of G_c which induces a clique we have that $|\mathcal{D} \cap C| = 1$. As a consequence $\mathcal{D} \setminus C$ is an independent dominating set of $G = (V, E)$ of size at most $k - 1$.

Thus, the problem of deciding whether a c-dense graph has an independent dominating set of size at most k is NP-complete. □

In the rest of this section, we provide an exponential time algorithm solving MIDS on c-dense graphs. The main idea of the algorithm is to find a large subset of vertices of large degree, to branch on these vertices and then to use the algorithm described in section 3.2.

Lemma 10. *For some fixed $1 \leq t \leq n$, $1 \leq t' \leq n-1$, any graph $G = (V, E)$ with $|E| \geq 1 + \frac{(t-1)(n-1) + (n-t+1)(t'-1)}{2}$ has a subset $T \subseteq V$ such that*

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

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

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 properties stated in the lemma. Then for any subset $T \subseteq V$ of size at least t, $\exists v \in T$ such that $d(v) < t'$. Then a such 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 11. *Every c-dense graph $G = (V, E)$ has a set $T \subseteq V$ such that*

(i) $|T| \geq 1 + n - \sqrt{2 - n + n^2 - 2cn^2}$,

(ii) for each vertex $v \in T$, $d(v) \geq n - \sqrt{2 - n + n^2 - 2cn^2}$.

Proof. We apply Lemma 10 with $t' = t - 1$. Since we have a dense graph, $|E| \geq cn^2$. Using inequality $1 + ((t-1)(n-1) + (n-t+1)(t-2))/2 \geq cn^2$ we obtain that in a dense graph the value of t in Lemma 10 is such that $1 + n - \sqrt{2 - n + n^2 - 2cn^2} \leq t \leq n \leq 1 + n + \sqrt{2 - n + n^2 - 2cn^2}$. □

The next theorem establishes that the independent dominating set algorithm for general graphs can be improved when the input graph has a large number of vertices of high degree.

Theorem 12. *Let $t > 0$ be a fixed integer. For any graph G on n vertices such that $|\{v \in V : d(v) \geq t-1\}| \geq t$, a $MIDS$ of G can be found in time $O(1.3575^{n-t})$.*

Proof. Let $t > 0$ be an integer and $G = (V, E)$ a graph fulfilling the condition of the theorem. Let $T = \{v \in V : d(v) \geq t-1\}$; thus $|T| \geq t$. Clearly, for every minimum independent dominating set $\mathcal{D}$ of G either at least one vertex of T belongs to the set $\mathcal{D}$, or none of the vertices in T belongs to $\mathcal{D}$, i.e. $T \cap \mathcal{D} = \emptyset$.

This permits to find a minimum independent dominating set of G using the following branching into two types of subproblems: "$v \in \mathcal{D}$" for each $v \in T$, and "$T \cap \mathcal{D} = \emptyset$". In both cases we shall apply the minimum independent dominating set algorithm of section 3.2 to solve the subproblem.

If you observe closely Theorem 5 of section 4, and particularly the part of the proof corresponding to the analysis of the running time, it is shown that the running time of our algorithm is $O(1.3575^{k(G)}) \leq O(1.3575^n)$ where $k(G)$ is a non standard measure on the size of G. Precisely, if $G = (F, M, E)$ is a marked graph, our algorithm can find a $MIDS$ (with respect to Definition 2) of G in time $O(1.3575^{|F|})$ since $k(G) \leq |F| \leq n$.

Consequently the running time for a subproblem will be $O(1.3575^{n-x})$, where x is the number of vertices eliminated from the original set of free vertices.

Consider now the two types of subproblems. Concerning the first one: for each vertex $v \in T$, we choose v in the minimum independent dominating set and we run the **ids** algorithm

presented in section 3.2 on an instance of size at most $n-(d(v)+1) \leq n-(t-1+1) = n-t$. Indeed, we remove from the set of vertices all vertices of $N[v]$. Concerning the second type of branching, we "discard the set T". In that case we have an instance of size at most $n - |T| = n - t$ since for every $v \in T$ we put v in the set of marked vertices. □

Theorem 13. `MIDS` *is solvable in time* $O(1.3575^{n\sqrt{1-2c}})$ *on c-dense graphs.*

Proof. Combining Theorem 12 and Lemma 11 we obtain an algorithm for solving the Minimum Independent Dominating Set problem in time

$$\begin{aligned} 1.3575^{n-(1+n-\sqrt{2-n+n^2-2cn^2})} &= 1.3575^{\sqrt{2-n+n^2-2cn^2}-1} \\ &\leq 1.3575^{\sqrt{2+n^2(1-2c)}} \\ &= O(1.3575^{n\sqrt{1-2c}}). \end{aligned}$$

□

7 Conclusions and Open Questions

In this paper we presented the first non trivial algorithm solving the minimum independent dominating set problem. Using a non standard measure on the size of the considered graph, we proved that our algorithm achieves a running time of $O(1.3575^n)$. Moreover we showed that $\Omega(1.3247^n)$ is a lower bound on the running time of this algorithm by exhibiting a family of graphs for which our algorithm has a high running time.

A natural question here is: is it is possible to obtain a better upper bound on the running time of the presented algorithm by considering another measure or using other techniques. Or is it possible that this upper bound is tight?

We have also provided a faster algorithm for INDEPENDENT DOMINATING SET on c-dense graphs. Moreover it is quite straightforward to use the technique of [13], a result of Alber and Niedermeier [1], and Theorem 12 to obtain an algorithm in time $O(1.3401^n)$ and exponential space for `MIDS` on circle graphs. For which other graph classes where the problem remains NP-complete can one design faster exponential-time algorithms?

References

[1] Alber, J. and Niedermeier, R. Improved Tree Decomposition Based Algorithms for Domination-like Problems, *Proceedings of LATIN 2002*, *LNCS* **2286**, (2002), pp. 613–628. 7

[2] Broersma, H., T. Kloks, D. Kratsch, and H. Müller, Independent sets in Asteroidal Triple-free graphs, *SIAM Journal on Discrete Mathematics*, **12**, (1999), pp. 276–287. 1

[3] Chang, M.-S., Efficient algorithms for the domination problems on interval and circular-arc graphs, *SIAM Journal on Computing*, **27**, (1998), pp. 1671–1694. 1

[4] Corneil, D.-G. and Y. Perl, Clustering and domination in perfect graphs, *Discrete Applied Mathematics*, **9**, (1984), pp. 27–39. 1

[5] Damian-Iordache, M. and S. V. Pemmaraju, Hardness of Approximating Independent Domination in Circle Graphs, *Proceedings of ISAAC 1999*, *LNCS* **1741**, (1999), pp. 56–69. 1

[6] Downey, R. G., Fellows, M. R., and McCartin, C., Parameterized Approximation Problems, *Proceedings of IWPEC 2006*, *LNCS* **4169**, (2006), pp. 121–129. 1

[7] Farber, M., Independent domination in chordal graphs, *Operation Research Letters*, **1**, (1982), pp. 134–138. 1

[8] Fomin, F. V., F. Grandoni, and D. Kratsch, Measure and Conquer: A Simple $O(2^{0.288n})$ Independent Set Algorithm, *Proceedings of SODA 2006*, (2006), pp. 18–25. 1

[9] Fomin, F. V., F. Grandoni, and D. Kratsch, Measure and conquer: Domination - A case study, *Proceedings of ICALP 2005*, *LNCS* **3380**, (2005), pp. 192–203. 1, 5

[10] 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. 1

[11] Fomin, F. V., D. Kratsch, and G. J. Woeginger, Exact (exponential) algorithms for the dominating set problem, *Proceedings of WG 2004*, *LNCS* **3353**, (2004), pp. 245–256. 1

[12] 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. 1

[13] Gaspers, S., D. Kratsch, and M. Liedloff, Exponential Time Algorithms for the Minimum Dominating Set Problem on Some Graph Classes, *Proceedings of SWAT 2006*, *LNCS* **4059**, (2006), pp. 148–159. 6, 7

[14] Grandoni, F., A note on the complexity of minimum dominating set, *Journal of Discrete Algorithms*, **4**, (2006), pp. 209–214. 1

[15] Halldórsson, M. M., Approximating the Minimum Maximal Independence Number, *Information Processing Letters*, **46**, (1993), pp. 169–172. 1

[16] Johnson, D. S., M. Yannakakis, and C. H. Papadimitriou, On generating all maximal independent sets, *Information Processing Letters*, **27**, (1988), pp. 119–123. 1

[17] Kratsch, D., and L. Stewart, Domination on Cocomparability Graphs, *SIAM Journal on Discrete Mathematics*, **6**, (1993), pp. 400–417. 1

[18] Moon, J. W., and L. Moser, On cliques in graphs, *Israel Journal of Mathematics*, **3**, (1965), pp. 23–28. 1

[19] Randerath, B., and I. Schiermeyer, Exact algorithms for Minimum Dominating Set, Technical Report zaik-469, Zentrum fur Angewandte Informatik, Köln, Germany, April 2004. 1

[20] Robson, J. M., Algorithms for maximum independent sets, *Journal of Algorithms*, **7**, (1986), pp. 425–440. 1

[21] Robson, J. M., Finding a maximum independent set in time $O(2^{n/4})$, Technical Report 1251-01, LaBRI, Université Bordeaux I, 2001. 1

[22] Schiermeyer, I., Problems remaining NP-complete for sparse or dense graphs, *Discussiones Mathematicae. Graph Theory* **15**, (1995), pp. 33–41. 6

[23] Tarjan, R. E., and A. E. Trojanowski, Finding a maximum independent set, *SIAM Journal on Computing*, **6**, (1977), pp. 537–546. 1

[24] Woeginger, G. J., Exact algorithms for NP-hard problems: A survey, *Combinatorial Optimization - Eureka, You Shrink!*, *LNCS* **2570**, (2003), pp. 185–207. 1