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

Serge Gaspers[1‡] and Mathieu Liedloff[2§]

[1] *Institute of Information Systems, Vienna University of Technology, Vienna, Austria.*
[2] *Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, Orléans, France.*

received 7th September 2010, revised 21st September 2011, accepted 6th January 2012.

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 neighbor in $\mathcal{D}$ and $\mathcal{D}$ is an independent set, i.e. no two vertices of $\mathcal{D}$ are adjacent in G. 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 $O(1.3569^n)$. 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.

Keywords: Exponential time algorithms, minimum independent dominating set, minimum maximal independent set

1 Introduction

During the last years the interest in the design of exact exponential time algorithms has grown significantly. Several nice surveys have been written on this subject. In Woeginger's first survey [39], he presents the major techniques used to design exact exponential time algorithms. We also refer the reader to the survey of Fomin et al. [16] discussing some more recent techniques for the design and the analysis of exponential time algorithms. In particular, they discuss Measure & Conquer and lower bounds.

In a graph $G = (V, E)$, a subset of vertices $S \subseteq V$ is *independent* if no two vertices of S share an edge, and S is *dominating* if every vertex from $V \setminus S$ has at least one neighbor in S. In the MAXIMUM INDEPENDENT SET problem (MIS), the input is a graph and the task is to find a largest independent set in this graph. In the MINIMUM DOMINATING SET problem (MDS), the input is a graph and the task is to find a smallest dominating set in this graph. A natural and well studied [1, 2, 8, 13, 26, 25] combination of these two problems asks for a subset of vertices of minimum cardinality that is both independent and dominating. This problem is called MINIMUM INDEPENDENT DOMINATING SET (MIDS). It is also known as MINIMUM MAXIMAL INDEPENDENT SET, since a vertex set is an independent dominating set if and only if it is a maximal independent set. Whereas there has been a lot of work on MIS and MDS in the field of exact algorithms, the best known exact algorithm for MIDS – prior to our work – trivially enumerates all maximal independent sets.

Known results. The MIS problem was among the first problems shown to be NP-hard [19]. It is known that a maximum independent set 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 that the number of maximal independent sets of a graph is upper bounded by $3^{n/3}$ [32] (see also [30]), and a result due to Johnson, Yannakakis and Papadimitriou, providing in [27] a polynomial delay algorithm to generate all maximal independent sets. Moreover many

† A preliminary version of this paper appeared in the *proceedings of WG 2006* [22].
‡ Email: `gaspers@kr.tuwien.ac.at`.
§ Email: `mathieu.liedloff@univ-orleans.fr`.

exact algorithms for this problem have been published, starting in 1977 by an $O(1.2600^n)$ algorithm by Tarjan and Trojanowski [36]. To date, the fastest known exponential space algorithms for MIS have been designed by Robson. His algorithm from 1986 [34] has running time $O(1.2108^n)$ and his unpublished computer-generated algorithm from 2001 [35] has running time $O(1.1889^n)$. Among the currently leading polynomial space algorithms, there is a very simple algorithm with running time $O(1.2210^n)$ by Fomin et al. [14, 17] from 2006, an $O(1.2132^n)$ time algorithm by Kneis et al. [28] from 2009, a very recent $O(1.2114^n)$ time algorithm by Bourgeois et al. [5], and Robson's unpublished $O(1.2025^n)$ time algorithm [35].

The MDS problem is also well known to be NP-hard [19]. Until 2004, 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 [18] Fomin et al. present an $O(1.9379^n)$ time algorithm, in [33] Randerath and Schiermeyer establish an $O(1.8899^n)$ time algorithm and Grandoni [24] obtains an $O(1.8026^n)$ time algorithm.

In 2005, Fomin et al. [15, 17] 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. Van Rooij and Bodlaender [37] further improved the polynomial-space algorithm to $O(1.5134^n)$ and the exponential-space algorithm to $O(1.5063^n)$. By now, the fastest published algorithm is due to van Rooij et al. In [38], they provide a $O(1.4969^n)$ time polynomial space algorithm.

It is known that a minimum independent dominating set (a mids, for short) can be found in polynomial time for several graph classes like interval graphs [7], chordal graphs [13], cocomparability graphs [29] and AT-free graphs [6], whereas the problem remains NP-complete for bipartite graphs [9] and comparability graphs [9]. Concerning approximation results, Halldórsson proved in [26] 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 [11].

The problem has also been considered in parameterized approximability. Downey et al. [12] have shown that it is $W[2]$-hard to approximate k-INDEPENDENT DOMINATING SET with a factor $g(k)$, for any computable function $g(k) \geq k$. In other words, unless $W[2] = FPT$, there is no algorithm with running time $f(k) \cdot n^{O(1)}$ (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 $g(k) \geq k$.

The first exponential time algorithm for MIDS has been observed by Randerath and Schiermeyer [33]. They use the result due to Moon and Moser [32] as explained previously and an algorithm enumerating all the maximal independent sets to obtain an $O(1.4423^n)$ time algorithm for MIDS. In 2006, an earlier conference version of this paper claimed an $O(1.3575^n)$ time algorithm [22]. However, a flaw concerning the main reduction rule was discovered by the authors and is repaired in the present paper. Several of the ideas introduced in [22] have been reused in recent work. Bourgeois et al. [4] designed a branch-and-reduce $O(1.3417^n)$ time algorithm for MIDS based on marked graphs and other concepts introduced in [22]. In [21], an algorithm is designed to count the number of disinct maximal independent sets in a graph, or equivalently the number of independent dominating sets, in time $O(1.3642^n)$. The general outline of that algorithm is similar to the one presented here, but several reduction and branching rules could not be used for the more general counting problem. The graph family used for the lower bound presented in Section 5 also makes an appearance in [21].

Our results. In this paper we present an $O(1.3569^n)$ time algorithm for solving MIDS using the Measure & Conquer approach to analyze its running time. As the bottleneck of the algorithm in [33] are the vertices of degree two, we develop several methods to handle them more efficiently such as marking some vertices and a reduction described in Subsection 3.1 to a constraint satisfaction problem. 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 a polynomial-space algorithm with running time $O(1.3569^n)$. 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 branching algorithms.

This paper is organized as follows. In Section 2, we introduce the necessary concepts and definitions.

Section 3 presents the algorithm for MIDS. 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 and we give a conclusion and open question in Section 6.

2 Preliminaries

Let $G = (V, E)$ be an undirected and simple graph. For a vertex $v \in V$ we denote by $N(v)$ the (open) neighborhood of v and by $N[v] = N(v) \cup \{v\}$ the closed neighborhood 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 neighbors in $V \setminus S$ of vertices in S and $N[S] = N(S) \cup S$. We also define $N_S(v)$ as $N(v) \cap S$, $N_S[v]$ 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 *bipartite* if V admits a partition into two independent sets. A bipartite graph $G = (V, E)$ is *complete bipartite* 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, a solution for the current problem instance is computed by recursing on smaller subinstances such that an optimal solution, if one exists, is computed for at least one subinstance. If the algorithm considers only one subinstance in a given case, we speak of a reduction rule, otherwise of a branching rule.

Consider a vertex $u \in V$ of degree two with two non-adjacent neighbors 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 the second branch considers all solution sets containing v_1. 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 such that $\mathcal{D}$ is an independent dominating set of the graph $(F \cup M, E)$.*

Remark 1 *It is possible that such an independent dominating set does not exist in a marked graph, for example if some marked vertex has no free neighbor.*

Finally, we introduce the notion of an *induced marked subgraph*.

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

Notions like neighborhood and degree in a marked graph (F, M, E) are the same as in the corresponding graph $(F \cup M, E)$.

3 Computing a mids on Marked Graphs

In this section we present an algorithm solving MIDS on a marked graph (F, M, E), assuming that no marked vertex has F-degree larger than 4. The algorithm is in fact not able to handle marked vertices with F-degree larger than 4 within our claimed running time bound. As the algorithm does not create new marked vertices with F-degree larger than 4, we may restrict the input instance of each recursion step to marked graphs where no marked vertex has F-degree larger than 4.

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 (non-marked) graphs as well. Also due to the definitions, edges incident to two marked vertices are irrelevant; throughout this paper we assume that there are no such edges.

Given a marked graph $G = (F, M, E)$, consider the graph $G[F]$ induced by its free vertices. In the following subsection we consider the special case when $G[F]$ is a disjoint union of cliques with some additional properties.

3.1 $G[F]$ is a disjoint union of cliques

Assume in this subsection that the graph $G[F]$ is a disjoint union of cliques with the following additional properties:

- each clique has size at most 4, and
- each marked vertex has at most 4 free neighbors.

We will arrive at such instances by branching on local structures that lead to favorable branching vectors. It is often the case in branching algorithms that vertices of high degree lead to good branching vectors, especially when the problem requires some kind of independence property of the solution. Moreover, instances with regular components often give the worst branching vectors and need to be handled with special care. For MIDS, this leads us to the core case where $G[F]$ is a disjoint union of cliques with the additional properties above, and for which branching seems a rather poor strategy.

We will transform this instance $G = (F, M, E)$ of MIDS into an instance (X, D, C) of the Constraint Satisfaction Problem (CSP). Let us briefly recall some definitions about CSP. Given a finite set $X = \{x_1, x_2, \ldots, x_n\}$ of n variables over domains $D(x_i)$, $1 \le i \le n$, and a set C of q constraints, CSP asks for an assignment of values to the variables, such that each variable is assigned a value from its domain, satisfying all the constraints. Formally, (d, p)-CSP is defined as follows:

Input: (X, D, C) where $X = \{x_1, x_2, \ldots, x_n\}$ is a set of variables over domains $D(x_i)$ of cardinality at most d, $1 \le i \le n$, and $C = \{c_1, c_2, ..., c_q\}$ is a set of constraints. Each constraint $c_i \in C$ is a pair $\langle t_i, R_i\rangle$ where $t_i = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_j}\rangle$ is a j-tuple of variables, with $j \le p$, and $R_i \subseteq D(x_{i_1}) \times D(x_{i_2}) \times \cdots \times D(x_{i_j})$.

Question: Is there a function f assigning to each variable $x_i \in X$, $1 \le i \le n$, a value of $D(x_i)$ such that for each constraint c_i, $1 \le i \le q$, $\langle f(x_{i_1}), ..., f(x_{i_j})\rangle \in R_i$?

Given a marked graph $G = (F, M, E)$ fulfilling the properties mentioned in the beginning of this subsection, we describe the construction of a $(4, 4)$-CSP instance. We label the cliques $K_1, K_2, \ldots, K_l$ of $G[F]$ respectively by $x_1, x_2, \ldots, x_l$. For each clique K_i, $1 \le i \le l$, label its vertices from v_i^1 to $v_i^{|K_i|}$. For each variable x_i, $1 \le i \le l$, we define its domain as $D(x_i) = \{1, 2, \ldots, |K_i|\}$.

Let $u_i \in M$ be a marked vertex and let $v_{i_1}^{k_1}, v_{i_2}^{k_2}, \ldots, v_{i_j}^{k_j}$ be the free neighbors of u_i. Thus, $j \le 4$. Let $t_i = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_j}\rangle$ be the j-tuple of variables corresponding respectively to the cliques containing $v_{i_1}^{k_1}, v_{i_2}^{k_2}, \ldots, v_{i_j}^{k_j}$. Let R_i be the set of all j-tuples $\langle w_{i_1}, w_{i_2}, \ldots, w_{i_j}\rangle$ over $D(x_{i_1}) \times D(x_{i_2}) \times \cdots \times D(x_{i_j})$ such that for at least one r, $1 \le r \le j$, the value of w_{i_r} is k_r and $\{u, v_{i_r}^{k_r}\}$ is an edge of the graph.

Finally, each marked vertex u_i leads to a constraint $\langle t_i, R_i\rangle$ of the set C. Due to the conditions on the given marked graph, the size of the domain of each variable is at most 4 and the number of variables involved in each constraint is at most 4.

Any independent dominating set of G contains exactly one vertex from each clique in $G[F]$. Selecting a vertex v_i^j in the MIDS instance corresponds to setting variable x_i to j in the CSP instance. The constraints make sure that for each marked vertex a free neighbor is selected. Thus, G has an independent dominating set if and only if the corresponding CSP instance has a solution.

We now use the following theorem of Angelsmark [3] showing that it is possible to restrict our attention to $(2, 4)$-CSP.

Theorem 4 (Theorem 11 of [3]) *If there exists a deterministic $O(\alpha^n)$ time algorithm for solving (e, p)-*CSP*, then for all $d > e$, there exists a deterministic $O((d/e+\epsilon)^n\alpha^n)$ time algorithm for solving (d, p)-*CSP*, for any $\epsilon > 0$.*

The constructive proof of this theorem shows how to transform a (d,p)-CSP instance on n variables into a set of (e,p)-CSP instances on at most n variables each, such that the (d,p)-CSP instance has a solution if and only if at least one of the (e,p)-CSP instances has a solution. The number of (e,p)-CSP instances of this construction is bounded by $\Pi_{i>e}(i/e+\epsilon)^{n_i} \leq (d/e+\epsilon)^n$, where n_i is the number of variables with domain size i in the (d,p)-CSP instance and $\epsilon > 0$ can be taken arbitrarily small.

We use this construction to transform our $(4,4)$-CSP instance into a set of $\Pi_{i>2}(i/2+\epsilon)^{N_i}$ $(2,4)$-CSP instances, where N_i is the number of cliques of size i in $G[F]$. Then, it is not hard to see that there exists a mids for G if and only if at least one of the $(2,4)$-CSP instances has an assignment of the variables which satisfies all the constraints of this CSP instance. Given a satisfying assignment f to such a CSP instance, the set $\bigcup_{i=1}^{l}\{v_i^{f(x_i)}\}$ is a solution to MIDS for G. We obtain the following theorem.

Theorem 5 *Let N_2, N_3 and N_4 be the number of variables (i.e. the number of cliques of $G[F]$) with domain size (resp. of size) 2, 3 and 4, respectively. The corresponding* CSP *instance can be solved in time $O((4/2+\epsilon)^{N_4} \cdot (3/2+\epsilon)^{N_3} \cdot \alpha^{N_4+N_3+N_2})$ where $O(\alpha^n)$ is the running time needed to solve a $(2,4)$-*CSP *instance on n variables, for any $\epsilon > 0$.*

The theorem can be combined with the following result of Moser and Scheder [31] providing an algorithm for solving $(2,4)$-CSP.

Theorem 6 ([31]) *Any $(2,4)$-*CSP *instance can be solved deterministically in time $O((1.5+\epsilon)^n)$, for any $\epsilon > 0$.*

Corollary 7 *Let $G = (F,M,E)$ be a marked graph such that $G[F]$ is a disjoint union of cliques of size at most 4, and each marked vertex has F-degree at most 4. Let N_i, $1 \leq i \leq 3$, be the number of free vertices with i free neighbors in G (thus $G[F]$ has N_i cliques of size $i+1$). A mids, if one exists, can be computed in time $O((1.5+\epsilon)^{N_1/2} \cdot (2.25+\epsilon)^{N_2/3} \cdot (3+\epsilon)^{N_3/4})$ or it can be decided within the same running time that the marked graph has no mids, for any $\epsilon > 0$.*

We remark that the procedure of Corollary 7 will not be a bottleneck in the final running time analysis of our algorithm, even if we use the $1.6^n \cdot n^{O(1)}$ by Dantsin et al. [10] to solve $(2,4)$-CSP instances instead of Theorem 6.

3.2 The Algorithm

In this subsection, we give Algorithm **ids** computing the size of a mids of a marked graph. Although the number of branching rules is quite large it is fairly simple to check that the algorithm computes the size of a mids (if one exists). It is also 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 the running time of Algorithm **ids**.

Once the algorithm has selected a vertex u, it makes recursive calls (that is, it branches) on subinstances of the current MIDS instance. There are different ways the algorithm branches and we give the most common ones now. The branching procedure $\textbf{branch\_one}(G,u)$ considers the two possibilities where u is in the solution set or where u is not in the solution set. In the recursive call corresponding to the second possibility, u is marked. The procedure returns

$$\min \begin{cases} 1 + \textbf{ids}(G[F \setminus N[u], M \setminus N(u)]); \\ \textbf{ids}(G[F \setminus \{u\}, M \cup \{u\}]). \end{cases}$$

The branching procedure $\textbf{branch\_all}(G,u)$ explores all possibilities in which u or a free neighbor of u is in the solution set. It returns

$$1 + \min_{v \in N_F[u]} \{\textbf{ids}(G[F \setminus N[v], M \setminus N(v)])\}.$$

The branching procedure $\textbf{branch\_mark}(G,u)$ additionally makes sure that the free neighbors of u are considered by increasing F-degree. Let $v_1, \ldots, v_{d_F(u)}$ denote the free neighbors of u, ordered by increasing F-degree. When considering the possibility that v_i is in the solution set, the procedure marks all vertices

v_j, $j < i$. It returns

$$1 + \min \begin{cases} \mathbf{ids}(G[F \setminus N[u], M \setminus N(u)]); \\ \min_{i=1..d_F(u)} \begin{cases} \mathbf{ids}(G[F \setminus (N[v_i] \cup \{v_1, \dots, v_{i-1}\}), \\ \qquad (M \cup \{v_1, \dots, v_{i-1}\}) \setminus N(v_i)]). \end{cases} \end{cases}$$

The branching procedure $\mathbf{branch\_all}$ is favored over $\mathbf{branch\_mark}$ if $\mathbf{branch\_mark}$ would create marked vertices of degree at least 5. Thus, starting with a graph where all the marked vertices have F-degree at most 4, Algorithm **ids** will keep this invariant. This property allows us to use the procedure described in the previous subsection whenever the graph induced by its free vertices is a collection of cliques of size at most 4. The correctness and running time analysis of **ids** are described in the next section.

4 Correctness and Analysis of the Algorithm

In our analysis, we assign so-called weights to free vertices. Free vertices having only marked neighbors can be handled without branching, as they are included in every independent dominating set (our algorithm simply postpones their treatment to the subroutine from Corollary 7). Hence, it is an advantage when the F-degree of a vertex decreases. The weights of the free vertices will therefore depend on 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 the marked graph G:

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

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

Theorem 8 *Algorithm* **ids** *solves* MIDS *in time* $O(1.3569^n)$.

Proof: Let $P[k]$ denote an upper bound on the running time of Algorithm **ids** on any instance of measure k, ignoring all polynomial factors in k. As each recursive call made by the algorithm is on an instance with at least one edge fewer and no reduction or branching rule increases k, $P[k]$ can be bounded by analyzing recurrences based on the measure of the created subinstances in those cases where the algorithm makes at least 2 recursive calls. We will analyze these cases one by one.

Case (1) A marked vertex that has no free neighbor cannot be dominated. Thus, such an instance has no independent dominating set.

Case (2) In this case, $G[F]$ is a disjoint union of cliques and u is a vertex from a clique of size $\ell \geq 6$ in $G[F]$. The branching $\mathbf{branch\_all}(G, u)$ creates ℓ subinstances whose measure is bounded by $k - \ell w_3$. The corresponding recurrence relation is $P[k] \leq \ell P[k - \ell w_3]$. For $\ell \geq 6$, the tightest of these recurrences is when $\ell = 6$:

$$P[k] \leq 6P[k - 6w_3]. \qquad \text{(rec. 1)}$$

Case (3) In this case, $G[F]$ is a disjoint union of cliques and u is a vertex from a clique of size 5 in $G[F]$. The branching $\mathbf{branch\_one}(G, u)$ creates 2 subinstances whose measure is bounded by $k - 5w_3$ and $k - w_3$, respectively. Note that the marked vertex which is created in the second branch has F-degree 4. The corresponding recurrence is

$$P[k] \leq P[k - 5w_3] + P[k - w_3]. \qquad \text{(rec. 2)}$$

```
Algorithm ids(G)
Input: A marked graph G = (F, M, E) with d_F(v) ≤ 4 for each v ∈ M.
Output: The size of a mids of G.

if ∃u ∈ M s.t. d_F(u) = 0 then
    return ∞;                                                        (1)
else if G[F] is a disjoint union of cliques then
    if ∃u ∈ F s.t. d_F(u) ≥ 5 then
        return branch_all(G, u);                                     (2)
    else if ∃u ∈ F s.t. d_F(u) = 4 then
        return branch_one(G, u);                                     (3)
    else
        return the solution determined by the algorithm of Corollary 7;   (4)
else if ∃u ∈ M s.t. d_F(u) = 1 then
    let v be the free neighbor of u
    return 1 + ids(G[F \ N[v], M \ N(v)]);                           (5)
else if ∃ a connected component B of G[F] s.t. |B| > 2 ∧ G[B] is complete bipartite then
    let B be partitioned into two independent sets X and Y
    return min { |X| + ids(G[F \ N[X], M \ N(X)]);
               { |Y| + ids(G[F \ N[Y], M \ N(Y)])     ;              (6)
else if ∃C ⊆ F s.t. |C| = 3 ∧ C is a clique ∧ ∃!v ∈ C s.t. d_F(v) ≥ 3 then
    return min{1 + ids(G[F \ N[v], M \ N(v)]); ids(G[F \ {v}, M])};  (7)
else
    choose u ∈ F such that
      (a) u is not contained in a connected component in G[F] that is a clique,
      (b) according to (a), u has minimum F-degree, and
      (c) according to (a) and (b), u has a neighbor in F of maximum F-degree.
    if d_F(u) = 1 then
        return branch_all(G, u);                                     (8)
    else if d_F(u) = 2 then
        if u has a neighbor of F-degree at most 4 then
            return branch_mark(G, u);                                (9)
        else
            return branch_all(G, u);                                 (10)
    else if d_F(u) = 3 then
        if all free neighbors of u have F-degree 3 then
            Let v ∈ N_F[u] such that G[N_F(v)] has at most 1 edge
            return branch_one(G, v);                                 (11)
        else if u has a neighbor v of F-degree 4 then
            return branch_one(G, v);                                 (12)
        else if u has a neighbor v of F-degree 5 then
                       { 1 + ids(G[F \ N[u], M \ N(u)]);
            return min { 1 + ids(G[F \ N[v], M \ N(v)]);     ;       (13)
                       { ids(G[F \ {u, v}, M ∪ {u, v}])
        else if u has two free neighbors of F-degree 3 then
            if N_F(u) is a clique then
                Let v_3 ∈ N_F(u) with maximum F-degree
                return min{1 + ids(G[F \ N[v_3], M \ N(v_3)]); ids(G[F \ {v_3}, M])};   (14)
            else
                return branch_mark(G, u);                            (15)
        else
            return branch_all(G, u);                                 (16)
    else if d_F(u) = 4 then
        return branch_one(G, u);                                     (17)
    else //  d_F(u) ≥ 5
        return branch_all(G, u);                                     (18)
```

Case (4) The graph induced by the free vertices is a disjoint union of cliques of size no more than 4. Corollary 7 is applied on the remaining marked graph and we note that the number n_i of vertices of F-degree i, $1 \leq i \leq 3$, in this graph is no more than $n_1 \leq k/w_1 \leq n/w_1$, $n_2 \leq k/w_2 \leq n/w_2$ and $n_3 \leq k/w_3 \leq n/w_3$ with $n_1 + n_2 + n_3 \leq n$.

Case (5) A marked vertex u with exactly one free neighbor v must be dominated by v. Thus, v is added to the mids and all its neighbors are deleted.

Case (6) 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 neighbors of either X or Y. The smallest possible subset B satisfying the conditions of this case is a P_3, that is a path on three vertices, as $|B| > 2$. Indeed, all smaller complete bipartite graphs are cliques and are handled by Case (4). Since we only count the number of free vertices, we obtain the following recurrence:

$$P[k] \leq 2P[k - 2w_1 - w_2]. \qquad \text{(rec. 3)}$$

It is clear that any complete bipartite component with more than three vertices leads to instances with smaller measures, which is less constraining for the running time.

Case (7) If there is a subset C of three free vertices which forms a clique and exactly one vertex $v \in C$ has free neighbors outside C, the algorithm either includes v in the solution set or it excludes v. In the first branch, all the neighbors of v are deleted (including C). In the second branch, note that v is not marked. Indeed, v's F-degree might be too high to be marked, and v's neighborhood contains a clique component in $G[F]$ of which one vertex is in every independent dominating set of the resulting marked graph, making the marking of v superfluous. We distinguish two cases based on the number of free neighbors of some free vertex $u \in N(v) \setminus C$.

1. Vertex u has one free neighbor (being v). In the first branch, $N[v]$ is deleted, and in the second branch, v is removed, u's F-degree decreases to 0, and the F-degree of both vertices in $C \setminus \{v\}$ decreases to 1. This gives the recurrence:

$$P[k] \leq P[k - w_1 - 2w_2 - w_3] + P[k + w_1 - 2w_2 - w_3]. \qquad \text{(rec. 4)}$$

2. Vertex u has F-degree at least 2. Then we obtain the recurrence:

$$P[k] \leq P[k - 3w_2 - w_3] + P[k + 2w_1 - 2w_2 - w_3]. \qquad \text{(rec. 5)}$$

Case (8) If there is a free vertex u with F-degree 1, a mids either includes u or its free neighbor v_1. Vertex v_1 cannot have F-degree one because this would contradict the first choice criterion (a) of u. For the analysis, we consider two cases:

1. $d_F(v_1) = 2$. Let x_1 denote the other free neighbor of v_1. Note that $d_F(x_1) \neq 1$ as this would have been handled by Case (6). We consider again two subcases:

   (a) $d_F(x_1) = 2$. When u is chosen in the independent dominating set, u and v_1 are deleted and the F-degree of x_1 decreases to one. When v_1 is chosen in the independent dominating set, u, v_1 and x_1 are deleted from the marked graph. So, we obtain the following recurrence for this case:

$$P[k] \leq P[k - 2w_2] + P[k - w_1 - 2w_2]. \qquad \text{(rec. 6)}$$

   (b) $d_F(x_1) \geq 3$. Vertices u and v_1 are deleted in the first branch, and u, v_1 and x_1 are deleted in the second branch. The recurrence for this subcase is:

$$P[k] \leq P[k - w_1 - w_2] + P[k - w_1 - w_2 - w_3]. \qquad \text{(rec. 7)}$$

2. $d_F(v_1) \geq 3$. At least one free neighbor of v_1 has F-degree at least 2, otherwise Case (6) applies since $N_F[v]$ is complete bipartite. Therefore the recurrence for this subcase is:

$$P[k] \leq P[k - w_1 - w_3] + P[k - 2w_1 - w_2 - w_3]. \qquad \text{(rec. 8)}$$

Case (9) If there is a free vertex u such that $d_F(u) = 2$ and u has a neighbor of F-degree at most 4 (as the neighbors v_1, v_2 of u are ordered by increasing F-degree, v_1 has F-degree at most 4), the algorithm uses $\mathbf{branch\_mark}(G, u)$ to branch into three subcases. Either u belongs to the mids, or v_1 is taken in the mids, or v_1 is marked and v_2 is taken in the mids. We distinguish three 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 (a cycle on 4 vertices) as this is a complete bipartite graph and would have been handled by Case (6). In the branches where u or v_1 belong to the mids, the three free vertices in $N[u]$ or $N[v_1]$ are deleted and two of their neighbors (T is a cycle on at least 5 vertices) have their F-degree reduced from 2 to 1. In the branch where v_1 is marked and v_2 is added to the mids, $N[v_2]$ is deleted and by Case (5), the other neighbor x_1 of v_1 is added to the mids, resulting in the deletion of $N[x_1]$ as well. In total, at least 5 free vertices of F-degree 2 are deleted in the third branch. Thus, we have the recurrence
$$P[k] \le 2P[k + 2w_1 - 5w_2] + P[k - 5w_2] \quad \text{(rec. 9)}$$
for this case.

2. $d_F(v_1) = 2, d_F(v_2) \ge 3$. The vertices v_1 and v_2 are not adjacent, otherwise Case (7) would apply. In the last branch, v_1 is marked and v_2 is added to the solution. If v_1 and v_2 have a common neighbor besides u, then the last branch is atomic because Case (1) applies as no vertex can dominate v_1. Otherwise, the reduction rule of Case (5) applies in the last branch and the other neighbor $x_1 \ne u$ is added to the solution as well. Thus, we have the recurrence
$$P[k] \le P[k - 2w_2 - w_3] + P[k - 3w_2] + P[k - 5w_2 - w_3]. \quad \text{(rec. 10)}$$

3. $3 \le d_F(v_1) \le 4$. We distinguish between two cases depending on whether there is an edge between v_1 and v_2.
   (a) v_1 and v_2 are not adjacent. Branching on u, v_1 and v_2 leads to the following recurrence:
   $$P[k] \le P[k - w_2 - 2w_3] + P[k - 3w_2 - w_3] + P[k - 3w_2 - 2w_3]. \quad \text{(rec. 11)}$$
   (b) v_1 and v_2 are adjacent. We distinguish two subcases depending on whether there is a vertex with F-degree 2 in $N_F^2(u)$, where $N_F^2(u)$ denotes the vertices at distance 2 from u in $G[F]$.
      i. There is a vertex with F-degree 2 in $N_F^2(u)$. Then,
      $$P[k] \le P[k + w_1 - 2w_2 - 2w_3] + 2P[k - 2w_2 - 2w_3]. \quad \text{(rec. 12)}$$
      ii. No vertex in $N_F^2(u)$ has F-degree 2. Then,
      $$P[k] \le P[k - w_2 - 2w_3] + 2P[k - w_2 - 3w_3]. \quad \text{(rec. 13)}$$

Case (10) If there is a free vertex u such that $d_F(u) = 2$ and none of the above cases apply, then v_1 and v_2 have degree at least 5 and the algorithm branches into the three subinstances of $\mathbf{branch\_all}(G, u)$: either u, v_1, or v_2 belongs to the mids, leading to the recurrence
$$P[k] \le P[k - w_2 - 2w_3] + 2P[k - 5w_2 - w_3]. \quad \text{(rec. 14)}$$

Case (11) If all neighbors of u have degree 3, then the connected component in $G[F]$ containing u is 3-regular due to the selection criteria of u. As (by criterion (a)) this component is not a clique, $N_F^2(u)$ is not empty. Thus, there exists some $v \in N_F[u]$ such that $G[N_F(v)]$ has at most one edge. This means that there are at least 4 edges with one endpoint in $N_F(v)$ and the other endpoint in $N_F^2(v)$. If $|N_F^2(v)| = 2$, the recurrence corresponding to the branching $\mathbf{branch\_one}(G, v)$ is
$$P[k] \le P[k + 2w_1 - 6w_3] + P[k + 3w_2 - 4w_3], \quad \text{(rec. 15)}$$
if $|N_F^2(v)| = 4$ it is
$$P[k] \le P[k + 4w_2 - 8w_3] + P[k + 3w_2 - 4w_3], \quad \text{(rec. 16)}$$

and if $|N_F^2(v)| = 3$ it is a mixture of the above two recurrences and is majorized by one or the other.

Case (12) If u has a neighbor v of F-degree 4, then the algorithm uses the branching procedure $\mathbf{branch\_one}(G, v)$. If v is taken in the mids, 5 vertices of degree at least 3 are removed from the instance. If v is marked, the F-degree of u decreases from 3 to 2. The corresponding recurrence is

$$P[k] \le P[k - 5w_3] + P[k + w_2 - 2w_3]. \qquad \text{(rec. 17)}$$

Case (13) If u has a neighbor v of F-degree 5, then the algorithm either takes u in the mids, or v, or it marks both u and v (note that v will have F-degree 4). The recurrence corresponding to this case is

$$P[k] \le P[k - 4w_3] + P[k - 6w_3] + P[k - 2w_3]. \qquad \text{(rec. 18)}$$

Case (14) In this case, $N_F[u]$ is a clique and v_3 is the only vertex from this clique that has free neighbors outside $N_F[u]$. The algorithm either takes v_3 in the mids or deletes it. Note that $N_F(v_3)$ includes a clique and that any mids of $G[F \setminus \{v_3\}, M]$ contains one vertex from this clique, which makes the marking of v_3 superfluous.

$$P[k] \le P[k - 7w_3] + P[k + 3w_2 - 4w_3]. \qquad \text{(rec. 19)}$$

Case (15) We distinguish two cases based on the neighborhood of v_3.

1. v_3 is adjacent to v_1 and v_2. Then, v_1 is not adjacent to v_2, otherwise Case (14) would apply. In the second branch, v_2's F-degree drops to 1 and in the third branch, v_1's neighbor in $N_F^2(u)$ is also selected by Case (5). This gives the recurrence

$$P[k] \le P[k - 4w_3] + P[k + w_1 - 5w_3] + P[k - 5w_3] + P[k - 7w_3]. \qquad \text{(rec. 20)}$$

2. v_3 is not adjacent to v_1 or to v_2. In the last branch, 7 vertices are deleted and one vertex is marked, giving

$$P[k] \le 3P[k - 4w_3] + P[k - 8w_3]. \qquad \text{(rec. 21)}$$

Case (16) In this case, u has at least two neighbors of degree at least 6. The recurrence corresponding to the branching $\mathbf{branch\_all}(G, u)$ is

$$P[k] \le 2P[k - 4w_3] + 2P[k - 7w_3]. \qquad \text{(rec. 22)}$$

Case (17) If u has degree 4, the algorithm branches along $\mathbf{branch\_one}(G, u)$, giving the recurrence

$$P[k] \le P[k - 5w_3] + P[k - w_3]. \qquad \text{(rec. 23)}$$

Case (18) If u has degree $\ell \ge 5$, the algorithm branches along $\mathbf{branch\_all}(G, u)$. The corresponding recurrence is $P[k] \le (\ell + 1)P[k - (\ell + 1)w_3]$, the tightest of which is obtained for $\ell = 5$:

$$P[k] \le 6P[k - 6w_3]. \qquad \text{(rec. 24)}$$

Finally the values of weights are computed with a convex optimization program [23] (see also [20]) to minimize the bound on the running time. Using the values $w_1 = 0.8482$ and $w_2 = 0.9685$ for the weights, one can easily verify that $P[k] = O(1.35684^k)$. In particular by this choice of the weights, the running-time required by Corollary 7 to solve the CSP instance whenever Case (4) is applied is no more than $O(1.3220^k)$ (it would be bounded by $O(1.3517^k)$ if we used the algorithm of Dantsin et al. [10] for solving (2,4)-CSP). Thus, Algorithm **ids** solves MIDS in time $O(1.3569^n)$. □

The tight recurrences of the latter proof (i.e. the worst case recurrences) are (rec. 13) and (rec. 16).

For the counting problem of determining the number of distinct independent dominating sets of a graph, Algorithm **ids** cannot be used due to the subroutine of Corollary 7, for which we are not aware of any counting version whose running time would not increase the overall running time of Algorithm **ids**. The counting problem can however be solved in time $O(1.3642^n)$ by an algorithm from [21] that is considerably simpler than Algorithm **ids**. Compared to the algorithm from [21], the present $O(1.3569^n)$ time algorithm is slightly faster due to improved branchings in a large number of cases and the deferral of an important

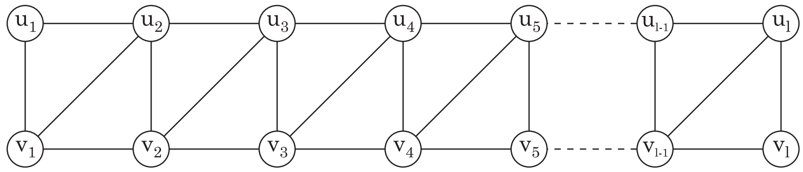

Fig. 1: graph G_l

case to the algorithm from Corollary 7. We believe that the main improvement from the $O(1.3417^n)$ time algorithm by Bourgeois et al. [4] is due to their analysis which also assigns weights to marked vertices. With all three branching algorithms having worst-case running time upper bounds differing by only a small margin, it seems that this kind of algorithms is hitting some kind of a barrier, and that completely new ideas will be necessary to obtain significant improvements. Indeed, in the next section we present a lower bound of $\Omega(1.3247^n)$ on the running time of Algorithm **ids**. The same lower bound holds for the algorithm from [21] (see [21] for the proof) and the algorithm from [4] (we do not give a proof here, but it is sufficient to go through the algorithm from [4] and adapt the proof from Section 5).

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 subinstances recursively processed by the algorithm, and consequently we obtained an upper bound on the worst case running time of Algorithm **ids**. However the use of another measure or a different method of analysis could perhaps provide a "better upper bound" without changing the algorithm but only improving the analysis.

How far is the given upper bound of Theorem 8 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 [15] (see also [17]) 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 9 *Algorithm* **ids** *solves* MIDS *in time* $\Omega(1.3247^n)$.

To prove Theorem 9 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) for some $l > 3$ defined by:

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

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 $MinDeg = \{u \in F : d_F(u) = \delta_F\}$ as the set of free vertices with smallest F-degree.

Denote the highest F-degree of the free neighbors of the vertices in $MinDeg$ by $\Delta_{\delta_F} = \max\big\{d_F(v) : v \in N_F(MinDeg)\big\}$.

Let $CandidateCase9 = \{u \in MinDeg : \exists v \in N_F(u) \text{ s.t. } d_F(v) = \Delta_{\delta_F}\}$ be the set of candidate vertices that **ids** can choose in Case (9). W.l.o.g. suppose that when $|CandidateCase9| \ge 2$ and **ids** would apply Case (9), it chooses the vertex with smallest index (e.g. if $CandidateCase9 = \{u_1, v_l\}$, the algorithm would choose u_1).

Lemma 10 *Let G'_l be the input of Algorithm* **ids**. *Suppose that* **ids** *only applies Case (9) in each recursive call (with respect to the previous rule for choosing a vertex). Then, in each call of* **ids** *where the remaining input graph has more than four vertices, one of the following two properties is fulfilled:*

(1) $CandidateCase9 = \{u_k, v_l\}$ for a certain k, $1 \le k \le l-2$, 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) $CandidateCase9 = \{v_k, v_l\}$ *for a certain* k, $1 \leq k \leq l-2$, *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 $CandidateCase9 = \{u_1, v_l\}$ for G'_l and that Property (1) is satisfied.

Suppose now that Property (1) is fulfilled. Then there exists an integer k, $1 \leq k \leq l-2$, such that $CandidateCase9 = \{u_k, v_l\}$. Since **ids** applies Case (9) respecting the rule for choosing the vertex in $CandidateCase9$, the algorithm chooses vertex u_k and branches on three subinstances:

(b1) Take u_k in the mids and remove $N[u_k]$. 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 subinstance, we obtain *Candi-dateCase*$9 = \{v_{k+1}, v_l\}$. 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 subinstance we obtain $CandidateCase9 = \{u_{k+2}, v_l\}$ 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 vertices are $\{v_{k+2}\} \cup \bigcup_{k+2<i\leq l}\{u_i, v_i\}$ and they are all free. For this remaining subinstance we obtain $CandidateCase9 = \{v_{k+2}, v_l\}$ and Property (2) is verified. (Note also that $|N[u_{k+1}] \cap \bigcup_{k\leq i\leq l}\{u_i, v_i\}| = 5$.)

Suppose now that Property (2) is fulfilled. Then there exists an integer k, $1 \leq k \leq l-2$, such that $CandidateCase9 = \{v_k, v_l\}$. Since **ids** applies Case (9) respecting the rule for choosing the vertex in $CandidateCase9$, the algorithm chooses vertex v_k and branches on three subinstances where it selects v_k, u_{k+1}, and v_{k+1} in the independent dominating set, creating subinstances satisfying Properties (1), (2), and (1), respectively, and containing 3, 4, and 5 vertices less than the parent instance, respectively. □

Now, we prove that, on input G_l, Algorithm **ids** applies Case (9) as long as the remaining graph has "enough" vertices.

Lemma 11 *Given the graph* G'_l *as input, as long as the remaining graph has more than four vertices, Algorithm* **ids** *applies Case (9) 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 none of Cases (1) to (8) can be applied. So, Case (9) is applied since $CandidateCase9 \neq \emptyset$ according to Lemma 10.

Consider now a graph obtained from G'_l by repeatedly branching using Case (9). By Lemma 10, the remaining graph has no marked vertices (this excludes that Cases (1) and (5) 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 excludes Cases (2)–(4)). The free vertices do not induce a bipartite graph since $\{v_{l-1}, u_l, v_l\}$ induces a C_3 (this excludes Case (6)). There is no clique C such that only one vertex of C has neighbors 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 neighbors outside the clique (this excludes Case (7)). Also, according to Lemma 10, the remaining graph has no vertex of degree 1 (this excludes Case (8)) and $CandidateCase9 \neq \emptyset$. Consequently, the algorithm applies Case (9). □

Figure 2 gives a part of the search tree illustrating the fact that our algorithm recursively branches on three subinstances with respect to Case (9).

Proof Proof of Theorem 9: Consider the graph G'_l and the search tree which results from branching using Case (9) until k vertices, $1 \leq k \leq 2l$, have been removed from the given input graph G'_l (G'_l has $2l$ vertices).

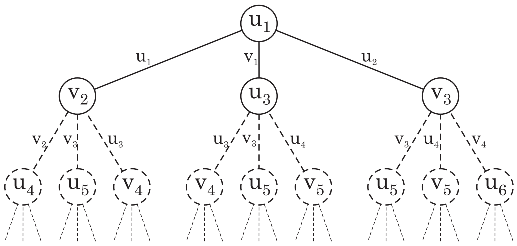

Fig. 2: a part of the search tree

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 10):

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

and therefore $L[k] \geq 1.3247^k$. Consequently, the maximum number of leaves that a search tree for **ids** can contain, given an input graph on n vertices, is $\Omega(1.3247^n)$. □

6 Conclusions and Open Questions

In this paper we presented a 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 $O(1.3569^n)$. 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?

References

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

[2] Allan, R. B. and R. Laskar, On domination and independent domination numbers of a graph, *Discrete Mathematics*, **23**, (1978), pp. 73–76.

[3] Angelsmark, O. Constructing Algorithms for Constraint Satisfaction and Related Problems : Methods and Applications, *PhD thesis*, *Linköping University, Sweden*, (2005).

[4] Bourgeois, M., B. Escoffier, and V. Th. Paschos, Fast Algorithms for min independent dominating set, *Proceedings of SIROCCO 2010*, *LNCS* **6058**, (2010), pp. 247–261.

[5] Bourgeois, M., B. Escoffier, V. Th. Paschos, and J. M. M. van Rooij, Fast algorithms for max independent set, *Algorithmica*, **62**, (2012), pp. 382–415.

[6] 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.

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

[8] Cockayne, E., Domination of undirected graphs – a survey, in *Theory and Applications of Graphs*, Springer, Berlin (1978), pp. 141–147.

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

[10] Dantsin E., A. Goerdt, E.A. Hirsch, R. Kannan, J.M. Kleinberg, C.H. Papadimitriou, P. Raghavan, and U. Schning, A deterministic (2-2/(k+1))n algorithm for k-SAT based on local search, *Theoretical Computer Science*, **289**, (2002), pp. 69–83.

[11] 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.

[12] Downey, R. G., Fellows, M. R., and McCartin, C., Parameterized Approximation Problems, *Proceedings of IW-PEC 2006*, *LNCS* **4169**, (2006), pp. 121–129.

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

[14] 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.

[15] 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.

[16] 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.

[17] Fomin, F. V., F. Grandoni, and D. Kratsch, A measure & conquer approach for the analysis of exact algorithms, *Journal of the ACM*, **56**, (2009).

[18] 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.

[19] 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.

[20] Gaspers, S., Exponential Time Algorithms: Structures, Measures, and Bounds, PhD thesis, University of Bergen, Norway, (2008).

[21] Gaspers, S., D. Kratsch, and M. Liedloff, On Independent Sets and Bicliques in Graphs, *Algorithmica*, **62**, (2012), pp. 637–658.

[22] Gaspers, S., and M. Liedloff, A Branch-and-Reduce Algorithm for Finding a Minimum Independent Dominating Set in Graphs, *Proceedings of WG 2006*, *LNCS* **4271**, (2006), pp. 78–89.

[23] Gaspers, S., and G. B. Sorkin, A universally fastest algorithm for Max 2-Sat, Max 2-CSP, and everything in between, *Journal of Computer and System Sciences*, **78**, (2012), pp. 305–335.

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

[25] Irving, R. W., On approximating the minimum independent dominating set, *Information Processing Letters*, **37**, (1991), pp. 197–200.

[26] Halldórsson, M. M., Approximating the minimum maximal independence number, *Information Processing Letters*, **46**, (1993), pp. 169–172.

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

[28] Kneis, J., A. Langer and P. Rossmanith, A Fine-grained Analysis of a Simple Independent Set Algorithm, *Proceedings of FSTTCS 2009*, *LIPIcs* **4**, (2009), pp. 287–298.

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

[30] Miller, R. E., and D. E. Muller, A problem of maximum consistent subsets, *IBM Research Report*, **RC-240**, J. T. Watson Research Center, Yorktown Heights, NY, 1960.

[31] Moser, R. A., and D. Scheder, A full derandomization of Schöning's k-SAT algorithm, *Proceedings of STOC 2011*, *ACM*, (2011), pp. 245–252.

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

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

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

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

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

[37] van Rooij, J. M. M., and H. L. Bodlaender, Exact algorithms for dominating set, *Discrete Applied Mathematics*, **159**, (2011), pp. 2147–2164.

[38] van Rooij, J. M. M., J. Nederlof, and T. C. van Dijk, Inclusion/Exclusion Meets Measure and Conquer, *Proceedings of ESA 2009*, *LNCS* **5757**, (2009), pp. 554–565.

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