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

Serge Gaspers† and Mathieu Liedloff‡§

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 \( D \) of a graph \( G = (V, E) \) is a subset of vertices such that every vertex in \( V \setminus D \) has at least one neighbor in \( D \) and \( D \) is an independent set, i.e. no two vertices of \( D \) 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

1 A preliminary version of this paper appeared in the proceedings of WG 2006 [22].
2 Email: gaspers@kr.tuwien.ac.at.
3 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 [33] 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 [55].

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 [8]. 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 distinct 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 $Ω(1.3247^n)$ 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 $N_S(v) = N[v] \cap S$, $N_S[v] = N[v] \cap S$, 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 $D$ of $G$ is a subset of free vertices such that $D$ 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 $D \subseteq V$ is a mids of a graph $G' = (V, E)$ if and only if $D$ 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 \( D(x_i), 1 \leq i \leq n \), 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 \( d \), \( 1 \leq i \leq n \), and \( C = \{c_1, c_2, \ldots, c_q\} \) is a set of constraints. Each constraint \( c_i \in C \) is a pair \((t_i, R_i)\) where \( t_i = \langle x_{i_1}, \ldots, x_{i_l} \rangle \) is a \( j \)-tuple of variables, with \( j \leq p \), 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 \( x_i \in X, 1 \leq i \leq n \), a value of \( D(x_i) \) such that for each constraint \( c_i, 1 \leq i \leq q \), \( (f(x_{i_1}), \ldots, f(x_{i_j})) \in R_i \) ?

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 \( K_i \), \( 1 \leq i \leq l \), label its vertices from \( v_{i_1}^k \) to \( v_{i_j}^k \). For each variable \( x_i \), \( 1 \leq i \leq l \), 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, \( j \leq 4 \). Let \( t_i = \langle x_{i_1}, \ldots, x_{i_j} \rangle \) 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 \((w_{i_1}, w_{i_2}, \ldots, w_{i_j})\) over \( D(x_{i_1}) \times D(x_{i_2}) \times \cdots \times D(x_{i_j}) \) such that for at least one \( r, 1 \leq r \leq j \), 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 \((t_i, R_i)\) 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 \( v_{i_r}^{k_r} \) 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 Angielski [32] showing that it is possible to restrict our attention to \((2, 4)\)-CSP.

**Theorem 4 (Theorem 11 of [32]):** If there exists a deterministic \( O(\alpha^n) \) time algorithm for solving \((c, p)\)-CSP, then for all \( d > c \), there exists a deterministic \( O((d/c + c)^n \alpha^n) \) time algorithm for solving \((d, p)\)-CSP, for any \( \epsilon > 0 \).
A branch-and-reduce algorithm for finding a minimum independent dominating set

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 \(\Pi_{i \geq \alpha}(i/e + \epsilon)^{N_i} \leq (d/e + \epsilon)^n\), 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 \(\Pi_{i \geq \alpha}(i/2 + \epsilon)^{N_i}\) \((2, 4)\)-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 \(\bigcup_{i=1}^{\tilde{f}^{(x)}} \{v_i^{(x)}\}\) 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 \(O\left((4/2 + \epsilon)^{N_4} \cdot (3/2 + \epsilon)^{N_3} \cdot (2 + \epsilon)^{N_2}/\alpha\right)\) 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 \(O((1.5 + \epsilon)^{N_4}/\alpha \cdot (2.25 + \epsilon)^{N_3}/\alpha \cdot (3 + \epsilon)^{N_2}/\alpha)\) 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 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

\[
\min \left\{ 1 + \text{ids}(G[F \setminus N[u], M \setminus N(u))\}; \right.
\]
\[\text{ids}(G[F \setminus \{u\}, M \cup \{u\}]).\]

The branching procedure branch_all\((G, u)\) explores all possibilities in which \(u\) or a free neighbor of \(u\) is in the solution set. It returns

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

The branching procedure 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 \(u_i\) is in the solution set, the procedure marks all vertices
in independent dominating set. Let

\[ P[k] \leq 6P[k - 6w_3]. \] (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 \texttt{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

\[ P[k] \leq P[k - 5w_3] + P[k - w_3]. \] (rec. 2)

The branching procedure \texttt{branch-all} is favored over \texttt{branch-mark} if \texttt{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 \texttt{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 \texttt{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 \texttt{ids} solves MIDS in time \( O(1.3569^n) \).

Proof: Let \( P[k] \) denote an upper bound on the running time of Algorithm \texttt{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 \texttt{branch-all}(G, u) creates \( \ell \) 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 \( \ell = 6 \):

\[ P[k] \leq 6P[k - 6w_3]. \] (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 \texttt{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

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

**Input:** A marked graph $G = (F, M, E)$ with $d_F(v) \leq 4$ for each $v \in M$.

**Output:** The size of a mids of $G$.

1. **if** $\exists u \in M$ s.t. $d_F(u) = 0$ **then**
   1. **return** $\infty$;
2. **else if** $G[F]$ is a disjoint union of cliques **then**
   1. **if** $\exists u \in F$ s.t. $d_F(u) \geq 5$ **then**
      1. **return** $\text{branch\_all}(G, u)$;
   2. **else if** $\exists u \in F$ s.t. $d_F(u) = 4$ **then**
      1. **return** $\text{branch\_one}(G, u)$;
   3. **else**
      1. **return** the solution determined by the algorithm of Corollary 7; \( \Box \)
3. **else if** $\exists u \in M$ s.t. $d_F(u) = 1$ **then**
   1. let $v$ be the free neighbor of $u$
   2. **return** $1 + \text{ids}(G[F \setminus N[v], M \setminus N(v)])$;
4. **else if** $\exists a$ connected component $B$ of $G[F]$ s.t. $|B| > 2$ \& $G[B]$ is complete bipartite **then**
   1. **let** $B$ be partitioned into two independent sets $X$ and $Y$
   2. **return** $\min \left\{ |X| + \text{ids}(G[F \setminus N[X], M \setminus N(X))| \right\}$;
5. **else if** $\exists C \subseteq F$ s.t. $|C| = 3$ \& $C$ is a clique \& $\exists v \in C$ s.t. $d_F(v) \geq 3$ **then**
   1. **return** $\min \{1 + \text{ids}(G[F \setminus N[v], M \setminus N(v)])| \text{ids}(G[F \setminus \{v\}, M])\}$;
6. **else**
   1. choose $u \in F$ such that
      1. (a) $u$ is not contained in a connected component in $G[F]$ that is a clique,
      2. (b) according to (a), $u$ has minimum $F$-degree, and
      3. (c) according to (a) and (b), $u$ has a neighbor in $F$ of maximum $F$-degree.
   2. **if** $d_F(u) = 1$ **then**
      1. **return** $\text{branch\_all}(G, u)$;
   3. **else if** $d_F(u) = 2$ **then**
      1. **if** $u$ has a neighbor of $F$-degree at most 4 **then**
         1. **return** $\text{branch\_mark}(G, u)$;
      2. **else**
         1. **return** $\text{branch\_all}(G, u)$;
   4. **else if** $d_F(u) = 3$ **then**
      1. **if** all free neighbors of $u$ have $F$-degree 3 **then**
         1. **let** $v \in N_F[u]$ such that $G[N_F(v)]$ has at most 1 edge
         2. **return** $\text{branch\_one}(G, v)$;
      2. **else if** $u$ has a neighbor $v$ of $F$-degree 4 **then**
         1. **return** $\text{branch\_one}(G, v)$;
      3. **else if** $u$ has a neighbor $v$ of $F$-degree 5 **then**
         1. **return** $\min \{1 + \text{ids}(G[F \setminus N[u], M \setminus N(u))| \right\}$;
         2. **else if** $u$ has two free neighbors of $F$-degree 3 **then**
            1. **let** $v_3 \in N_F(u)$ with maximum $F$-degree
            2. **return** $\min \{1 + \text{ids}(G[F \setminus N[v_3], M \setminus N(v_3))| \text{ids}(G[F \setminus \{v_3\}, M])\}$;
      4. **else if** $d_F(u)$ is a clique **then**
         1. **return** $\text{branch\_mark}(G, u)$;
      5. **else**
         1. **return** $\text{branch\_all}(G, u)$;
   5. **else if** $d_F(u) = 4$ **then**
      1. **return** $\text{branch\_one}(G, u)$;
   6. **else**
      1. **return** $\text{branch\_all}(G, u)$;

**Notes:**

- $\Box$ indicates a placeholder for further details.
- The algorithm handles various cases based on the degree and structure of the graph $G$.
- The `branch_all` function is used for branching when no specific criteria are met.
- The `branch_one` function is used when a single choice is made.
- The `branch_mark` function marks nodes for further processing.

**Corollary:**

The solution determined by the algorithm of Corollary 7 provides an optimal minimum independent dominating set for the graph $G$. 

---

**References:**

The algorithm is based on a branch-and-reduce approach, where the graph is partitioned and decisions are made based on the degree and structure of the graph. The objective is to minimize the size of the independent dominating set through a recursive decision-making process.
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 $v$ 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:

$$P[k] \leq 2P[k-2w_1-w_2].$$

(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:

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

(rec. 4)

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

$$P[k] \leq P[k-3w_2-w_3] + P[k+2w_1-2w_2-w_3].$$

(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 $v_1$ 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:

$$P[k] \leq P[k-2w_2] + P[k-w_1-2w_2].$$

(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:

$$P[k] \leq P[k-w_1-w_2] + P[k-w_1-w_2-w_3].$$

(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:

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

(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 $\text{branch\_mark}(G, u)$ to branch into three subcases. Either $u$ belongs to the mids, or $v_1$ is taken in the mids, or $v_2$ 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

$$P[k] \leq 2P[k + 2w_1 - 5w_2] + P[k - 5w_2] \quad \text{(rec. 9)}$$

for this case.

2. $d_F(v_1) = 2, d_F(v_2) \geq 3$. 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 $x_1 \neq u$ is added to the solution as well. Thus, we have the recurrence

$$P[k] \leq P[k - 2w_2 - w_3] + P[k - 3w_2] + P[k - 5w_2 - w_3]. \quad \text{(rec. 10)}$$

3. $3 \leq d_F(v_1) \leq 4$. 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:

$$P[k] \leq P[k - w_2 - 2w_3] + P[k - 3w_2 - w_3] + P[k - 3w_2 - 2w_3]. \quad \text{(rec. 11)}$$

(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,

$$P[k] \leq P[k + w_1 - 2w_2 - 2w_3] + 2P[k - 2w_2 - 2w_3]. \quad \text{(rec. 12)}$$

ii. No vertex in $N_F^2(u)$ has $F$-degree 2. Then,

$$P[k] \leq P[k - w_2 - 3w_3] + 2P[k - w_2 - 3w_3]. \quad \text{(rec. 13)}$$

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 $\text{branch\_all}(G, u)$: either $u, v_1$, or $v_2$ belongs to the mids, leading to the recurrence

$$P[k] \leq P[k - w_2 - 2w_3] + 2P[k - 5w_2 - w_3]. \quad \text{(rec. 14)}$$

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 $\text{branch\_one}(G, v)$ is

$$P[k] \leq P[k + 2w_1 - 6w_3] + P[k + 3w_2 - 4w_3]. \quad \text{(rec. 15)}$$

if $|N_F^2(v)| = 4$ it is

$$P[k] \leq P[k + 4w_2 - 8w_3] + P[k + 3w_2 - 4w_3]. \quad \text{(rec. 16)}$$
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 \texttt{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

\[
P[k] \leq P[k - 5w_3] + P[k + w_2 - 2w_3].
\]

(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

\[
P[k] \leq P[k - 4w_3] + P[k - 6w_3] + P[k - 2w_3].
\]

(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.

\[
P[k] \leq P[k - 7w_3] + P[k + 3w_2 - 4w_3].
\]

(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

\[
P[k] \leq P[k - 4w_3] + P[k + w_1 - 5w_3] + P[k - 5w_3] + P[k - 7w_3].
\]

(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

\[
P[k] \leq 3P[k - 4w_3] + P[k - 8w_3].
\]

(rec. 21)

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

\[
P[k] \leq 2P[k - 4w_3] + 2P[k - 7w_3].
\]

(rec. 22)

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

\[
P[k] \leq P[k - 5w_3] + P[k - w_3].
\]

(rec. 23)

**Case (18)** If \(u\) has degree \(\ell \geq 5\), the algorithm branches along \texttt{branch\_all}(\(G, u\)). The corresponding recurrence is

\[
P[k] \leq (\ell + 1)P[k - (\ell + 1)w_3],
\]

the tightest of which is obtained for \(\ell = 5\):

\[
P[k] \leq 6P[k - 6w_3].
\]

(rec. 24)

Finally the values of weights are computed with a convex optimization program \texttt{(23)} (see also \texttt{(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 \texttt{(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. \texttt{(10)} for solving (2,4)-CSP). Thus, Algorithm \texttt{ids} solves MIDS in time \(O(1.3569^{n})\).

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

For the counting problem of determining the number of distinct independent dominating sets of a graph, Algorithm \texttt{ids} cannot be used due to the subroutine of Corollary \texttt{(7)} for which we are not aware of any counting version whose running time would not increase the overall running time of Algorithm \texttt{ids}. The counting problem can however be solved in time \(O(1.3642^{n})\) by an algorithm from \texttt{(21)} that is considerably simpler than Algorithm \texttt{ids}. Compared to the algorithm from \texttt{(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
Fomin et al. obtain a bound that gives a really good estimation on the precision of the analysis. For example, in [15] (see also [17]), perhaps provide a “better upper bound” without changing the algorithm but only improving the analysis.

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:

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

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

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

Let $\text{CandidateCase9} = \{u \in \text{MinDeg} : \exists v \in N_F(u) \text{ s.t. } d_F(v) = \Delta_{\text{MinDeg}}\}$ be the set of candidate vertices that **ids** can choose in Case (9). W.l.o.g. suppose that when $|\text{CandidateCase9}| \geq 2$ and **ids** would apply Case (9), it chooses the vertex with smallest index (e.g. if $\text{CandidateCase9} = \{u_1, v_1\}$, 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. $\text{CandidateCase9} = \{u_k, v_1\}$ for a certain $k$, $1 \leq k \leq l - 2$, and
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 \( G'_1 \), it is clear that none of Cases (1) to (8) can be applied. So, Case (9) is applied since \( \text{CandidateCase}9 = \emptyset \) 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 \( v_1 \) (this excludes Cases (2)–(4)). The free vertices do not induce a bipartite graph since \( \{v_{l-1}, u_1, v_1\} \) 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 \( \text{CandidateCase}9 \neq \emptyset \). 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 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).
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 $\text{ids}$ 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


