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

Serge Gaspers[1] and Mathieu Liedloff[2]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway, `serge.gaspers@ii.uib.no`

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

Abstract. A dominating set $\mathcal{D}$ of a graph $G = (V, E)$ is a subset of vertices such that every vertex in $V \setminus \mathcal{D}$ has at least one neighbour in $\mathcal{D}$. Moreover if $\mathcal{D}$ is an independent set, i.e. no vertices in $\mathcal{D}$ are pairwise adjacent, then $\mathcal{D}$ is said to be an independent dominating set. Finding a minimum independent dominating set in a graph is an NP-hard problem. We give an algorithm computing a minimum independent dominating set of a graph on n vertices in time $O(1.3575^n)$. Furthermore, we show that $\Omega(1.3247^n)$ is a lower bound on the worst-case running time of this algorithm.

1 Introduction

During the last years the interest in the design of exact exponential time algorithms has been growing significantly. Nice surveys have been written on this subject. In one due to Woeginger [18], the author emphasizes the major techniques used to design exact exponential time algorithms. We also refer the reader to the recent survey of Fomin et al. [7] discussing some new techniques in the design of exponential time algorithms. In particular they discuss Measure & Conquer and lower bounds.

The Minimum Independent Dominating Set problem (`MIDS`) is also known as Minimum Maximal Independent Set, since every independent dominating set is a maximal independent set. This problem asks for a set of minimum cardinality that is both independent and dominating. Whereas Maximum Independent Set and Minimum Dominating Set have been studied very deeply in the field of exact algorithms, the best known exact algorithm for `MIDS` trivially enumerates all maximal independent sets.

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

It is known that a `MIS` of a graph on n vertices can be computed in $O(1.4423^n)$ time by combining a result due to Moon and Moser, who showed in 1965 [13] that the number of maximal independent sets of a graph is upper bounded by

$3^{n/3}$, and a result due to Johnson, Yannakakis and Papadimitriou, providing in [11] a polynomial delay algorithm to generate all maximal independent sets. Moreover many exact algorithms for this problem have been published, starting in 1977 by an $O(1.2600^n)$ algorithm by Tarjan and Trojanowski [17]. The best known algorithms for MIS until now are an $O(1.2108^n)$ algorithm by Robson [15] in 1986, a very long algorithm of running time $O(1.1889^n)$ by Robson [16] in 2001 and a very simple algorithm with running time $O(1.2210^n)$ by Fomin et al. [5] in 2006.

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

Until recently, the only known exact exponential time algorithm to solve MDS asked for trivially enumerating the 2^n subsets of vertices. The year 2004 saw a particular interest in providing some faster algorithms for solving this problem. Indeed, three papers with exact algorithms for MDS were published. In [8] Fomin et al. present an $O(1.9379^n)$ time algorithm, in [14] Randerath and Schiermeyer establish an $O(1.8899^n)$ time algorithm and Grandoni [9] obtains an $O(1.8026^n)$ time algorithm.

By now, the fastest published algorithm is due to Fomin et al. [6]. They use the Measure & Conquer approach to obtain an algorithm with running time $O(1.5263^n)$ and using polynomial space. By applying a memorization technique they show that this running time can be reduced to $O(1.5137^n)$ when allowing exponential space usage.

A natural and well studied combination of these two problems asks for a subset of vertices of minimum cardinality that is both dominating and independent. This problem is called Minimum Independent Dominating Set (MIDS).

It has been established that a minimum independent dominating set ($MIDS$) can be found in polynomial time for several graph classes like interval graphs [2], chordal graphs [4], cocomparability graphs [12] and AT-free graphs [1], whereas the problem remains NP-complete for bipartite graphs [3] and comparability graphs [3]. Concerning inapproximability results, Halldórsson established in [10] that there is no constant $\epsilon > 0$ such that MIDS can be approximated within a factor of $n^{1-\epsilon}$ in polynomial time, assuming $P \neq NP$.

To the best of our knowledge, the only paper giving an exact exponential time algorithm for MIDS has been written by Randerath and Schiermeyer [14]. They use the result due to Moon and Moser [13] as explained previously and an algorithm enumerating all the maximal independent sets to obtain an $O(1.4423^n)$ time algorithm for MIDS.

Our results. In this paper we present an $O(1.3575^n)$ time algorithm for solving MIDS using the Measure & Conquer approach to analyze its running time. As the bottleneck of the algorithm in [14] are the vertices of degree two, we use some nice tricks to handle them more efficiently such as marking some vertices and a sophisticated reduction rule described in section 3.1. Combined with some elaborated branching rules, this enables us to lower bound shrewdly the progress made by the algorithm at each branching step, and thus to obtain an algorithm which improves the best known result from $O(1.4423^n)$ to $O(1.3575^n)$. Furthermore, we obtain a very close lower bound of $\Omega(1.3247^n)$ on the running time of our algorithm, which is very rare for non trivial exponential time algorithms.

2 Preliminaries

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

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

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

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

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

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

Remark 1. It is possible that such an independent dominating set does not exist in a marked graph, namely if a marked vertex has no free neighbours.

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

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

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

3 Computing a *MIDS* on Marked Graphs

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

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

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

3.1 Eliminating Cliques in $G[F]$

Consider the function **RedClique**. (See next page.)

Lemma 1. *Let $G = (F, M, E)$ be a marked graph and C a connected component of $G[F]$ which is a clique. The function* **RedClique** *computes in polynomial time a marked graph $G' = (F', M', E')$ such that:*

(i) the size of a MIDS of G is equal to the size of a MIDS of G' plus one,
(ii) $F' = F \setminus C$,
(iii) no edge of $E' - E$ has both end points in F', i.e. the function adds no edge between two free vertices.

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

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

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

Assume now that $|C| \geq 2$ and that every vertex in C has at least one neighbour in M. Then, the function will create one new marked vertex $h_{i,j}$ for every two vertices $h_i, h_j \in N(C)$ that do not share a same neighbour in C. It replaces $N[C]$ by these new marked vertices. A vertex $h_{i,j}$ will be adjacent to a vertex $v \in F \setminus C$ iff h_i or h_j was adjacent to v. So, when all vertices $h_{i,j}$ will be dominated by vertices in $F \setminus C$ in G', at least all the vertices in $N(C)$ except the

```
Function RedClique($G = (F, M, E), C \subseteq F$)
Input: A marked graph $G = (F, M, E)$ and a clique $C \subseteq F$ such that $C$ is
       a connected component of $G[F]$.
Output: A marked graph $G' = (F', M', E')$ s.t. $G'$ has the properties
        defined in Lemma 1.
  if $|C| = 1$ then
      $G' \leftarrow G[F \setminus C, M \setminus N(C)]$;
  else
      if $\exists v \in C$ s.t. $N_M(v) = \emptyset$ then
          $G' \leftarrow$ RedClique($G[F - \{v\}, M], C - \{v\}$)
      else
          let $N(C) = \{h_1, h_2, \dots, h_k\}$
          $H \leftarrow \emptyset$
          for $i \leftarrow 1$ to $k - 1$ do
              for $j \leftarrow i + 1$ to $k$ do
                  if $N_C(h_i) \cap N_C(h_j) = \emptyset$ then
                      add to $H$ a new marked vertex $h_{i,j}$
          $G' = (F', M', E') \leftarrow G[F \setminus C, M \setminus N(C)]$
          $M' \leftarrow M' \cup H$
          foreach $h_{i,j} \in H$ do
              foreach $v \in N_F(N[C])$ s.t. $\{v, h_i\} \in E$ or $\{v, h_j\} \in E$ do
                  $E' \leftarrow E' \cup \{v, h_{i,j}\}$
  return $G'$
```

neighbours of a unique vertex $u \in C$ are dominated in G. It is then clear which vertex of C will be in the $MIDS$. And whenever a vertex $h_{i,j}$ is not dominated in G', no vertex of C can dominate all undominated vertices in $N(C)$ in G.

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

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

3.2 The Algorithm

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

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

4 Correctness and Analysis of the Algorithm

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

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

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

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

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

Theorem 1. *Algorithm* **ids** *solves the minimum independent dominating set problem in time* $O(1.3575^n)$.

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

case (0) If the set of vertices is empty, the algorithm returns 0 since it has computed an independent dominating set of the marked graph.

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

case (2) If there is a marked vertex u with only one free neighbour v, the only possibility for u to be dominated is to add v to the $MIDS$. So, $N[v]$ is deleted and the measure k decreases at least by $2w_1$.

case (3) If there is a clique C of free vertices which are not adjacent to any other free vertices, we use the function of Lemma 1 to remove C. Since the number of free vertices decreases by $|C|$ and no new edges are added between any two free vertices, the F-degrees of the remaining free vertices do not increase. Thus the measure k decreases. (Note that the number of marked vertices and their F-degree can increase by this reduction, but these parameters do not occur in our measure.)

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

(3). Since we only count the number of free vertices, we obtain the following recurrence:

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

It is clear that any complete bipartite component with more than three vertices would lead to a better recurrence.

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

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

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

1. $d_F(v) = 2$. Let x denote the other free neighbour of v. Note that $d_F(x) \neq 1$ as this would have been handled by case (4). We consider again two subcases:
   (a) $d_F(x) = 2$. When u is chosen in the independent dominating set, u and v are deleted and the degree of x decreases to one. When v is chosen in the independent dominating set, u, v and x are deleted from the marked graph. So, we obtain the following recurrence for this case:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$P[k] \leq P[k + (2-q)w_1 - (4-q)w_2 - w_3] + P[k + w_1 - (4-q)w_2 - (1+q)w_3] + P[k - 2w_2 - 2w_3]. \quad (11)$$

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

$$P[k] \leq P[k - w_2 - 2w_3] + P[k - 3w_2 - w_3] + P[k - 3w_2 - 2w_3]. \quad (12)$$

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

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

$$P[k] \leq P[k + w_1 - 2w_2 - 2w_3] + 2P[k - 2w_2 - 2w_3]. \quad (13)$$

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

$$P[k] \leq P[k + w_1 - 2w_2 - 2w_3] + P[k + w_1 - 2w_2 - 3w_3] + P[k - 2w_2 - 2w_3]. \quad (14)$$

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

$$P[k] \leq P[k - w_2 - 2w_3] + 2P[k - w_2 - 3w_3]. \quad (15)$$

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

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

$$P[k] \leq P[k + 2w_2 - 6w_3] + P[k + 3w_2 - 4w_3]. \quad (16)$$

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

$$P[k] \leq P[k - 5w_3] + P[k - w_3]. \quad (17)$$

Finally the values of weights are computed by a random local search for minimizing the bound on the running time. Using the values $w_1 = 0.8372$ and $w_2 = 0.9644$ for the weights, one can easily verify that $P[k] = O(1.3575^n)$. □

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

5 A Lower Bound on the Running Time of the Algorithm

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

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 [6] Fomin et al. obtain a $O(1.5137^n)$ 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 2. *Algorithm* **ids** *solves the minimum independent dominating set problem in* $\Omega(1.3247^n)$.

Proof. Due to space restriction we only give a sketch of the proof. A detailed proof will be given in the full version of this paper.

Consider the graph $G_n = (V_n, E_n)$ (see Fig. 1) defined by $V_n = \{u_i, v_i : 1 \leq i \leq n\}$ and $E_n = \{u_1, v_1\} \cup \big\{\{u_i, v_i\}, \{u_i, u_{i-1}\}, \{v_i, v_{i-1}\}, \{u_i, v_{i-1}\} : 2 \leq i \leq n\big\}$. We denote by $G'_n = (V, \emptyset, E)$ the marked graph corresponding to the graph $G_n = (V, E)$.

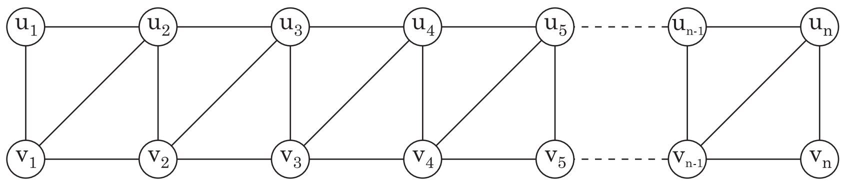

Fig. 1. graph G_n

It is possible to show that given the graph G'_n as input, as long as the remaining graph has more than four vertices, algorithm **ids** applies case (7) in each recursive call. Moreover, without loss of generality, we can suppose that whenever **ids** would apply case (7), it chooses the vertex with smallest index.

Consider now the graph G'_n and the search tree which results of branchings using case (7) until k vertices, $1 \leq k \leq 2n$, have been removed from the given input graph G'_n (G'_n has $2n$ vertices). Denote by $L[k]$ the number of leaves in this search tree. It is not hard to see that this would lead to the recurrence $L(k) = L(k-3) + L(k-4) + L(k-5)$ and therefore $L(k) \geq 1.3247^k$. Consequently 1.3247^n is a lower bound of the maximum number of leaves that a search tree for **ids** could give with an input graph on n vertices. $\square$

References

1. Broersma, H., T. Kloks, D. Kratsch, and H. Müller, Independent sets in Asteroidal Triple-free graphs, *SIAM J. Discrete Math.*, **12**, (1999), pp. 276–287.
2. Chang, M.-S., Efficient algorithms for the domination problems on interval and circular-arc graphs, *SIAM J. Comput.*, **27**, (1998), pp. 1671–1694.
3. Corneil, D.-G. and Y. Perl, Clustering and domination in perfect graphs, *Discrete. Appl. Math.*, **9**, (1984), pp. 27–39.
4. Farber, M., Independent domination in chordal graphs, *Operation Research Letters*, **1**, (1982), pp. 134–138.
5. 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.
6. 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.
7. 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.
8. 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.
9. Grandoni, F., A note on the complexity of minimum dominating set, *J. Discrete Algorithms*, **4**, (2006), pp. 209–214.
10. Halldórsson, M. M., Approximating the Minimum Maximal Independence Number, *Inf. Process. Lett.*, **46**, (1993), pp. 169–172.
11. Johnson, D. S., M. Yannakakis, and C. H. Papadimitriou, On generating all maximal independent sets, *Inf. Process. Lett.*, **27**, (1988), pp. 119–123.
12. Kratsch, D., and L. Stewart, Domination on Cocomparability Graphs, *SIAM J. Discrete Math.*, **6**, (1993), pp. 400–417.
13. Moon, J. W., and L. Moser, On cliques in graphs, *Israel J. Math.*, **3**, (1965), pp. 23–28.
14. Randerath, B., and I. Schiermeyer, Exact algorithms for Minimum Dominating Set, Technical Report zaik-469, Zentrum fur Angewandte Informatik, Köln, Germany, April 2004.
15. Robson, J. M., Algorithms for maximum independent sets, *J. Algorithms*, **7**, (1986), pp. 425–440.
16. Robson, J. M., Finding a maximum independent set in time $O(2^{n/4})$, Technical Report 1251-01, LaBRI, Université Bordeaux I, 2001.
17. Tarjan, R. E., and A. E. Trojanowski, Finding a maximum independent set, *SIAM J. Comput.*, **6**, (1977), pp. 537–546.
18. Woeginger, G. J., Exact algorithms for NP-hard problems: A survey, *Combinatorial Optimization - Eureka, You Shrink!*, *LNCS* **2570**, (2003), pp. 185–207.