REPORTS IN INFORMATICS

ISSN 0333-3590

Measure and Conquer: Domination – A Case Study

Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch

REPORT NO 294 April 2005

Department of Informatics
UNIVERSITY OF BERGEN
Bergen, Norway

This report has URL `http://www.ii.uib.no/publikasjoner/texrap/ps/2005-294.ps`

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

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

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

Measure and Conquer: Domination – A Case Study

Fedor V. Fomin*
Department of Informatics
University of Bergen
N-5020 Bergen, Norway
fomin@ii.uib.no

Fabrizio Grandoni†
Dipartimento di Informatica
Università di Roma "La Sapienza"
Via Salaria 113
00198 Roma, Italy
grandoni@di.uniroma1.it

Dieter Kratsch
LITA, Université de Metz
57045 Metz Cedex 01, France
kratsch@sciences.univ-metz.fr

Abstract

Davis-Putnam-style exponential-time backtracking algorithms are the most common algorithms used for finding exact solutions of NP-hard problems. The analysis of such recursive algorithms is based on the bounded search tree technique: a measure of the size of the subproblems is defined; this measure is used to lower bound the progress made by the algorithm at each branching step.

For the last 30 years the research on exact algorithms has been mainly focused on the design of more and more sophisticated algorithms. However, measures used in the analysis of backtracking algorithms are usually very simple. In this paper we stress that a more careful choice of the measure can lead to significantly better worst case time analysis.

As an example, we consider the minimum dominating set problem. The currently fastest algorithm for this problem has running time $O(2^{0.850n})$ on n-nodes graphs. By measuring the progress of the (same) algorithm in a different way, we refine the time bound to $O(2^{0.598n})$. A good choice of the measure can provide such a (surprisingly big) improvement; this suggests that the running time of many other exponential-time recursive algorithms is largely overestimated because of a "bad" choice of the measure.

Keywords: Algorithms and data structures, exponential-time exact algorithm, NP-hard problem, dominating set

1 Introduction

The interest in exact and fast exponential-time algorithms solving hard problems dates back to the sixties and seventies [13, 25]. The last decade has led to much

*Supported by Norges forskningsråd project 160778/V30.
†Supported by Web-Minds project of the Italian Ministry of University and Research, under the FIRB program.

research in fast exponential-time algorithms. Examples of recently developed exponential algorithms are algorithms for Maximum Independent Set [15, 23], (Maximum) Satisfiability [4, 14, 17, 19, 24, 26], Coloring [2, 3, 6], Treewidth [8], and many others (see the recent survey written by Woeginger [27] for an overview).

Most of the currently fastest exact algorithms for NP-hard problems are recursive algorithms. In order to bound the total number of subproblems generated by such algorithms, the *bounded search tree* technique is often used: one defines a suitable *measure* of the size of the subproblems. This measure is used to lower bound the "progress" made by the algorithm at each branching step.

Though the algorithms considered may be rather complicated, the measures used in their analysis are usually very simple. In this paper we remark that a more careful choice of the measure can lead to much tighter time bounds.

In order to show that, we consider one of the best known NP-hard problems: the *minimum dominating set* problem. The currently fastest exact algorithm for this problem is a recursive algorithm of running time $\mathcal{O}^*(2^{0.850n})$ on n-nodes graphs [10, 11][1]. Here we present a refined analysis, based on a different measure of the size of the subproblems generated and show that the same algorithm has indeed running time $\mathcal{O}^*(2^{0.598n})$. This surprisingly big improvement suggests the possibility that the running times of many other exponential-time recursive algorithms (including possibly the one presented here) are largely overestimated because of a "bad" choice of the measure in their analysis. Despite the importance of the problem, only few works address this issue [2, 7].

Since the current tools do not seem to be strong enough to support an analysis of exponential-time recursive algorithms providing tight running time upper bounds, it is natural to ask for lower bounds (notice that we are concerned with lower bounds on the complexity of a particular algorithm and not with lower bounds on the complexity of an algorithmic problem). A lower bound may give an idea of how far the analysis is from being tight. There are several results known on lower exponential bounds for different branching algorithms for SAT (see e.g. [1, 18]) but we are not aware of lower bounds for existing exponential-time recursive graph algorithms. One of the reasons to this could be that for most of the graph problems the construction of good lower bounds is often difficult even for very simple algorithms. In this paper we prove a $\Omega(2^{0.333n})$ lower bound on the time complexity of our minimum dominating set algorithm. The large gap between the upper bound and the lower bound suggests the possibility that the analysis of the algorithm can be further refined (possibly by measuring the size of the subproblems in a smarter way).

Previous results on dominating set. The *minimum dominating set* problem (`MDS`) is a classic NP-hard graph optimization problem which fits into the broader class of *domination* and *covering* problems on which hundreds of papers have been written; see e.g. the survey [12] by Haynes et al. The dominating set problem is also one of the basic problems in parameterized complexity [5]; it is W[2]-complete and thus it is unlikely that the problem is fixed parameter tractable. What are the best time complexities for the dominating set problem in n-node graphs $G = (V, E)$ that we can possibly hope for? It has been observed in [9] that (unless some very unexpected things happen in computational complexity theory) there is no sub-exponential time (i.e. of running time $c^{o(n)}$ for some constant c) algorithm solving dominating set problem. There is the trivial $O^*(2^n)$ algorithm that simply searches through all the 2^n subsets of V. Hence, we can only hope for time complexities of the form $O^*(2^{cn})$, with some small value $c < 1$. Although `MDS` is a natural and

[1]Throughout this paper we use a modified big-Oh notation that suppresses all polynomially bounded factors. For functions f and g we write $f(n) = \mathcal{O}^*(g(n))$ if $f(n) = \mathcal{O}(g(n)poly(n))$, where $poly(n)$ is a polynomial.

very interesting problem concerning the design and analysis of exponential-time algorithms, no exact algorithm for MDS faster than the trivial one had been known until very recently. In 2004 three different sets of authors seemingly independently published algorithms breaking the trivial "2^n-barrier". The algorithm of Fomin et al. [9] uses a deep graph-theoretic result due to Reed [21], providing an upper bound on the domination number of graphs of minimum degree three. The most time consuming part of their algorithm is an enumeration of all subsets of nodes of cardinality at most $3n/8$, thus the overall running time is $\mathcal{O}^*(2^{0.955n})$. The algorithm of Randerath and Schiermeyer [20] uses a very nice and cute idea (including matching techniques) to restrict the search space. The most time consuming part of their algorithm enumerates all subsets of nodes of cardinality at most $n/3$, thus the overall running time is $\mathcal{O}^*(2^{0.919n})$. Finally, the fastest algorithm known prior to our work is due to Grandoni [10, 11], who described a $\mathcal{O}^*(2^{0.850n})$ algorithm for MDS.

Our Results. We show that MDS can be solved in $\mathcal{O}^*(2^{0.610n})$ time using polynomial space. The running time of our algorithm can be reduced at the cost of exponential space to $\mathcal{O}^*(2^{0.598n})$ which is a significant improvement of all known results on MDS. To solve the problem we represent MDS as a set cover problem which allows us to use a search tree based algorithm. This idea was first used in [10, 11]. To obtain running time $\mathcal{O}^*(2^{0.610n})$ we do not add more and more sophisticated rules to existing algorithms which is a usual practice to improve on the exponential base. Instead we give a simple and easy to implement algorithm and observe how the careful choice of the measure changes the algorithm analysis dramatically. Our refined analysis leads to a multivariate recurrence. For a general treatment of this type of recurrences we refer to Eppstein's paper [7]. Since the analysis of our search tree based algorithms is so depended on the choice of the measure, it is natural to ask for (exponential) lower bounds on the running time of the algorithm. We prove that our algorithm requires $\Omega(2^{0.333n})$ steps.

2 Definitions and Basic Algorithm

Let $G = (V, E)$ be an n-node undirected, simple graph without loops. The open *neighborhood* of a node v is denoted by $N(v) = \{u \in V : uv \in E\}$, and the closed neighborhood of v is denoted by $N[v] = N(v) \cup \{v\}$. A set $A \subseteq E$ of edges of $G = (V, E)$ is an *edge cover*, if every node of G is incident to an edge of A; the edge set A is a *matching* if no node of G is incident to two edges of A.

The minimum dominating set problem. Let $G = (V, E)$ be a graph. A set $D \subseteq V$ is called a *dominating set* for G if every node of G is either in D, or adjacent to some node in D. The *domination number* $\gamma(G)$ of a graph G is the minimum cardinality of a dominating set of G. The *Minimum Dominating Set* problem (MDS) asks to determine $\gamma(G)$.

The minimum set cover problem. In the *Minimum Set Cover* problem (MSC) we are given a universe $\mathcal{U}$ of elements and a collection $\mathcal{S}$ of (non-empty) subsets of $\mathcal{U}$. The aim is to determine the minimum cardinality of a subset $\mathcal{S}' \subseteq \mathcal{S}$ which *covers* $\mathcal{U}$, that is such that

$$\cup_{S \in \mathcal{S}'} S = \mathcal{U}.$$

The *frequency* of $u \in \mathcal{U}$ is the number of subsets $S \in \mathcal{S}$ in which u is contained. For the sake of simplicity, we always assume in this paper that $\mathcal{S}$ covers $\mathcal{U}$:

$$\mathcal{U} = \mathcal{U}(\mathcal{S}) \triangleq \cup_{S \in \mathcal{S}} S.$$

With this assumption, an instance of MSC is univocally specified by $\mathcal{S}$.

Figure 1 A recursive algorithm for minimum set cover.

```
1  int msc(𝒮) {
2      if(|𝒮| = 0) return 0;
3      if(∃S, R ∈ 𝒮 : S ⊆ R) return msc(𝒮\{S});
4      if(∃u ∈ 𝒰(𝒮)∃ a unique S ∈ 𝒮 : u ∈ S) return 1+msc(del(S, 𝒮));
5      take S ∈ 𝒮 of maximum cardinality;
6      if(|S| = 2) return poly-msc(𝒮)
7      return min{msc(𝒮\{S}), 1+msc(del(S, 𝒮))};
8  }
```

We recall that, if all the subsets of $\mathcal{S}$ are of cardinality two, `MSC` can be solved in polynomial time via the following standard reduction to maximum matching. Consider the graph $\tilde{G}$ which has a node u for each $u \in \mathcal{U}$, and an edge uv for each subset $S = \{u, v\}$ in $\mathcal{S}$. Thus we have to compute a minimum edge cover of $\tilde{G}$. To compute a minimum edge cover of $\tilde{G}$ we compute a maximum matching M in $\tilde{G}$. Then, for each unmatched node u, we add to M an arbitrary edge incident to u (if no such edge exists, there is no set cover at all). The subsets corresponding to M form a minimum set cover.

`MDS` can be naturally reduced to `MSC` by imposing $\mathcal{U} = V$ and $\mathcal{S} = \{N[v]|\ v \in V\}$. Note that $N[v]$ is the set of nodes dominated by v, thus D is a dominating set of G if and only if $\{N[v]|\ v \in D\}$ is a set cover of $\{N[v]|\ v \in V\}$. Thus every minimum set cover of $\{N[v]|\ v \in V\}$ corresponds to a minimum dominating set of G.

At first view such a transformation from one NP-hard problem to another seems to be completely useless: The only known exact algorithms for `MSC` are brute force $\mathcal{O}^*(2^{|\mathcal{S}|})$ and $\mathcal{O}^*(2^{|\mathcal{U}|})$ [9] dynamic programming algorithms. Both algorithms result in an $\mathcal{O}^*(2^n)$ algorithm for `MDS` and it seems that such an approach is not interesting. Not at all! On second thought the transformation from `MDS` to `MSC` becomes very helpful. It enables the use of a search tree based algorithm to solve `MSC`, and thus also `MDS`.

Basic algorithm. We consider a simple recursive algorithm `msc` for solving `MSC`. The algorithm is a slight modification of the algorithm from [11] and it makes use of the following observation.

Lemma 2.1. *For a given* `MSC` *instance* $\mathcal{S}$*:*

1. *If there are two distinct sets S and R in $\mathcal{S}$, $S \subseteq R$, then there is a minimum set cover which does not contain S.*
2. *If there is an element u of $\mathcal{U}$ which belongs to a unique $S \in \mathcal{S}$, then S belongs to every set cover.*

Note that each subset of cardinality one satisfies exactly one of the properties in Lemma 2.1.

A basic version of `msc` is described in Figure 1. If $|\mathcal{S}| = 0$ (line 2), $msc(\mathcal{S}) = 0$. Otherwise (lines 3 and 4), the algorithm *tries* to reduce the size of the problem without branching, by applying one of the Properties 1 and 2 of Lemma 2.1. Specifically, if there are two sets S and R, $S \subseteq R$, we have $\texttt{msc}(\mathcal{S}) = \texttt{msc}(\mathcal{S} \backslash S)$. If there is an element u which is contained in a unique set S, we have $\texttt{msc}(\mathcal{S}) = 1 + \texttt{msc}(del(S, \mathcal{S}))$, where $del(S, \mathcal{S}) = \{Z | Z = R \backslash S \neq \emptyset, R \in \mathcal{S}\}$ is the instance of `MSC` which is obtained from $\mathcal{S}$ by removing the elements of S from the subsets in $\mathcal{S}$, and by eventually removing the empty sets obtained.

If none of the two properties above applies, the algorithm takes (line 5) a set $S \in \mathcal{S}$ of maximum cardinality. If $|S| = 2$ (line 6), the algorithm directly solves the problem with the polynomial time algorithm `poly-msc` based on the reduction to maximum matching. Otherwise (line 7), it branches on the two subproblems $S_{IN} = del(S, \mathcal{S})$ (the case where S belongs to the minimum set cover) and $S_{OUT} = \mathcal{S} \backslash S$ (corresponding to the case S is not in the minimum set cover). Thus

$$\texttt{msc}(\mathcal{S}) = \min\{\texttt{msc}(\mathcal{S} \backslash \{S\}), 1 + \texttt{msc}(del(S, \mathcal{S}))\}.$$

Notice that with simple modifications, the algorithm can also provide one minimum set cover (besides its cardinality).

To emphasize the importance of the measure we sketch the analysis of the algorithm with a simple measure (taken from [11]). Let us choose the following *measure* $k(\mathcal{S}')$ of the size of a `MSC` instance $\mathcal{S}'$,

$$k(\mathcal{S}') = |\mathcal{S}'| + |\mathcal{U}(\mathcal{S}')|.$$

Let $\ell(k)$ be the number of leaves in the search tree generated by the algorithm to solve a problem of size $k = k(\mathcal{S})$. If one of the conditions of lines 3 and 4 is satisfied, $\ell(k) \leq \ell(k-1)$. Let S be the set selected in line 5. If $|S| = 2$, the algorithm directly solves the problem in polynomial time ($\ell(k) = 1$). Otherwise ($|S| \geq 3$), the algorithm branches on the two subproblems $\mathcal{S}_{OUT} = \mathcal{S} \backslash \{S\}$ and $\mathcal{S}_{IN} = del(\mathcal{S}, S)$. The size of $\mathcal{S}_{OUT}$ is $k-1$ (one set removed from $\mathcal{S}$). The size of $\mathcal{S}_{IN}$ is at most $k-4$ (one set removed from $\mathcal{S}$ and at least three elements removed from $\mathcal{U}$). This brings us to $\ell(k) \leq \ell(k-1) + \ell(k-4)$. We conclude that $\ell(k) \leq \alpha^k$, where $\alpha = 1.3802\ldots < 1.3803$ is the (unique) positive root of the polynomial $(x^4 - x^3 - 1)$. It turns out that the total number of subproblems solved is within a polynomial factor from $\ell(k)$. Moreover, solving each subproblem takes polynomial time. Thus the complexity of the algorithm is $\mathcal{O}^*(\ell(k)) = \mathcal{O}^*(\alpha^k) = \mathcal{O}^*(1.3803^{|\mathcal{S}|+|\mathcal{U}|}) = \mathcal{O}^*(2^{0.465(|\mathcal{S}|+|\mathcal{U}|)})$.

In next section we will show how to refine the running time analysis to $\mathcal{O}^*(2^{0.305(|\mathcal{S}|+|\mathcal{U}|)})$ via a more careful choice of the measure $k(\mathcal{S}')$ (without modifying the algorithm!).

3 Refined Analysis

In this section we show that algorithm `msc` has time complexity $\mathcal{O}^*(2^{0.305(|\mathcal{S}|+|\mathcal{U}|)})$.

Our result is based on the following observation. Removing a large set has a different impact on the "progress" of the algorithm than removing a small one. In fact, when we remove a large set, we decrease the frequency of many elements. Decreasing elements frequency pays of on long term, since the elements of frequency one can be filtered out (without branching). A dual argument holds for the elements. Removing an element of high frequency is somehow preferable to removing an element of small frequency. In fact, when we remove an element occurring in many sets, we decrease the cardinality of all such sets by one. This is good on long term, since sets of cardinality one can be filtered out. Both phenomena are not taken into account in the measure used in [10]. With that measure, by removing one set (element), we decrease the size of the problem by one, no matter which is the cardinality (frequency) of the set (element) considered.

This suggests the idea to give a different "weight" to sets of different cardinality and to elements of different frequency. In particular, let n_i denote the number of subsets $S \in \mathcal{S}$ of cardinality i. Let moreover m_j denote the number of elements $u \in \mathcal{U}$ of frequency j. We will use the following measure $k = k(\mathcal{S})$ of the size of $\mathcal{S}$:

$$k(\mathcal{S}) = \sum_{i \geq 1} w_i \, n_i + \sum_{j \geq 1} v_j \, m_j,$$

where the weights $w_i, v_j \in (0, 1]$ will be fixed in the following. Note that $k \leq$

$|\mathcal{S}| + |\mathcal{U}|$. The quantities

$$\Delta w_i = \begin{cases} w_i - w_{i-1} & \text{if } i \geq 3, \\ w_2 & \text{if } i = 2, \end{cases} \quad \text{and} \quad \Delta v_i = \begin{cases} v_i - v_{i-1} & \text{if } i \geq 3, \\ v_2 & \text{if } i = 2, \end{cases}$$

turn out to be useful in the analysis. Intuitively, Δw_i (Δv_i) is the reduction of the size of the problem corresponding to the reduction of the cardinality of a set (of the frequency of an element) from i to $i-1$. Note that this holds also in the case $i = 2$. In fact, in that case the size of the problem first increases by $1 - w_2$ ($1 - v_2$), but the new set of cardinality one (the new element of frequency one) introduced is removed before the next branching, with a reduction of the size by one. Thus one has an overall reduction by $1 - (1 - w_2) = w_2$ ($1 - (1 - v_2) = v_2$).

Theorem 3.1. *Algorithm* `msc` *solves* `MSC` *in time* $\mathcal{O}^*(2^{0.305(|\mathcal{U}|+|\mathcal{S}|)})$.

Proof. The correctness of the algorithm is trivial. In order to simplify the running time analysis, we will make the following assumptions:

- $w_1 = v_1 = 1$ and $w_i = v_i = 1$ for $i \geq 6$;
- $0 \leq \Delta w_i \leq \Delta w_{i-1}$ for $i \geq 2$.

Note that this implies $w_i \geq w_{i-1}$ for every $i \geq 3$ (excluding sets of cardinality one, larger sets have larger weights). Moreover, $\Delta w_i = \Delta v_i = 0$ for $i \geq 7$.

Let $P_h(k)$ be the number of subproblems of size h, $0 \leq h \leq k$, solved by `msc` to solve a problem of size k. Clearly, $P_k(k) = 1$. Consider the case $h < k$ (which implies $|\mathcal{S}| \neq 0$). If one of the condition of lines 3 and 4 holds, one set S is removed from $\mathcal{S}$. Thus the reduction of the size of the problem is at least w_2 (corresponding to the case $|S| = 2$) and $P_h(k) \leq P_h(k - w_2)$. Otherwise, let S be the subset selected in line 5. If $|S| = 2$, no subproblem is generated ($P_h(k) = 0$). Otherwise ($|S| \geq 3$), `msc` generates two subproblems $\mathcal{S}_{IN} = del(S, \mathcal{S})$ and $\mathcal{S}_{OUT} = \mathcal{S} \backslash S$.

Consider the subproblem $\mathcal{S}_{OUT}$. The size of $\mathcal{S}_{OUT}$ decreases by $w_{|S|}$ because of the removal of S. Let r_i be the number of elements of S of frequency i. Note that there cannot be elements of frequency 1. Consider an element $u \in S$ of frequency $i \geq 2$. When we remove S, the frequency of u decreases by one. As a consequence, the size of the subproblem decreases by Δv_i. Thus the overall reduction of the size of $\mathcal{S}_{OUT}$ due to the reduction of the frequencies is at least

$$\sum_{i \geq 2} r_i \, \Delta v_i = \sum_{i=2}^{6} r_i \, \Delta v_i.$$

Suppose that there is an element $u \in S$ of frequency 2. Let $R \neq S$ be the other set containing u. When we remove S, we have to include R in the set cover. Thus we reduce the size of the problem by at least w_2 (corresponding to the case $|R| = 2$). Also $R \setminus S$ is not empty (otherwise condition of line 3 of the algorithm is met) and thus all elements of $R \setminus S$ are removed when we include R in the set cover. This reduces the size by at least v_2 (corresponding to the case that the frequency of z is 2). Therefore the overall reduction of the size of $\mathcal{S}_{OUT}$ due to the removal of the sets R is at least $r_2 \, w_2 + \delta(r_2) \, v_2$, where $\delta(r_2) = 0$ for $r_2 = 0$, and $\delta(r_2) = 1$ otherwise.

Consider now the subproblem $\mathcal{S}_{IN}$. The size of $\mathcal{S}_{IN}$ decreases by $w_{|S|}$ because of the removal of S. Consider an element $u \in S$ of frequency i ($i \geq 2$). The size of $\mathcal{S}_{IN}$ further decreases by v_i because of the removal of u. Thus the overall reduction due to the removal of the elements u of S is

$$\sum_{i \geq 2} r_i \, v_i = \sum_{i=2}^{6} r_i \, v_i + r_{\geq 7},$$

where $r_{\geq i}$ is the number of elements of S of frequency at least i. Let R be a set sharing an element u with S. Note that $|R| \leq |S|$. By removing u, the cardinality of R is reduced by one. This implies a reduction of the size of $\mathcal{S}_{IN}$ by $\Delta\, w_{|R|} \geq \Delta\, w_{|S|}$. Thus the overall reduction of $\mathcal{S}_{IN}$ due to the reduction of the cardinalities of the sets R is at least:

$$\Delta\, w_{|S|} \sum_{i \geq 2} (i-1)\, r_i \geq \Delta\, w_{|S|} \left(\sum_{i=2}^{6} (i-1)\, r_i + 6 \cdot r_{\geq 7} \right).$$

Note that this quantity is 0 for $|S| \geq 7$. Putting all together, for all the possible values of $|S| \geq 3$ and of the r_i such that

$$\sum_{i=2}^{6} r_i + r_{\geq 7} = |S|,$$

we have the following set of recursions

$$P_h(k) \leq P_h(k - \Delta\, k_{OUT}) + P_h(k - \Delta\, k_{IN}),$$

where

- $\Delta\, k_{OUT} \triangleq w_{|S|} + \sum_{i=2}^{6} r_i\, \Delta\, v_i + r_2\, w_2 + \delta(r_2)\, v_2$,
- $\Delta\, k_{IN} \triangleq w_{|S|} + \sum_{i=2}^{6} r_i\, v_i + r_{\geq 7} + \Delta\, w_{|S|} \left(\sum_{i=2}^{6} (i-1)\, r_i + 6 \cdot r_{\geq 7} \right)$.

Since $\Delta\, w_{|S|} = 0$ for $|S| \geq 7$, we have that each recurrence with $|S| \geq 8$ is "dominated" by some recurrence with $|S| = 7$. For this reason, we restrict our attention only to the cases $3 \leq |S| \leq 7$. Thus we consider a large but finite number of recurrences. For every fixed 8-tuple $(w_2, w_3, w_4, w_5, v_2, v_3, v_4, v_5)$ the number $P_h(k)$ is upper bounded by α^{k-h}, where α is the largest number from the set of real roots of the set of equations

$$\alpha^k = \alpha^{k - \Delta\, k_{OUT}} + \alpha^{k - \Delta\, k_{IN}}$$

corresponding to different combinations of values $|S|$ and r_i. Thus the estimation of $P_h(k)$ boils up to choosing the weights minimizing α. This optimization problem is interesting in its own and we refer to Eppstein's work [7] on quasi-convex programming for general treatment of such problems.

We numerically obtained the following values of the weights:

$$w_i = \begin{cases} 0.3774 & \text{if } i = 2, \\ 0.7548 & \text{if } i = 3, \\ 0.9095 & \text{if } i = 4, \\ 0.9764 & \text{if } i = 5, \end{cases} \quad \text{and} \quad v_i = \begin{cases} 0.3996 & \text{if } i = 2, \\ 0.7677 & \text{if } i = 3, \\ 0.9300 & \text{if } i = 4, \\ 0.9856 & \text{if } i = 5, \end{cases}$$

which yields $\alpha \leq 1.2352\ldots < 1.2353$. In Table 1 the values of $|S|$ and r_i of the eight worst case recurrences are listed.

Let K denote the set of the possible sizes of the subproblems solved. Note that $|K|$ is polynomially bounded. The total number $P(k)$ of subproblems solved satisfies:

$$P(k) \leq \sum_{h \in K} P_h(k) \leq \sum_{h \in K} \alpha^{k-h} \leq |K| \alpha^k.$$

The cost of solving a problem of size $h \leq k$, excluding the cost of solving the corresponding subproblems (if any), is a polynomial $poly(k)$ of k. Thus the time complexity of the algorithm is

$$\mathcal{O}^*(poly(k)|K|\alpha^k) = \mathcal{O}^*(1.2353^{|\mathcal{U}|+|\mathcal{S}|}) = \mathcal{O}^*(2^{0.305(|\mathcal{U}|+|\mathcal{S}|)}).$$

Table 1 The eight worst case recurrences.

$\lvert S\rvert$	$(r_2, r_3, r_4, r_5, r_6, r_{\geq 7})$
6	$(0, 0, 0, 0, 0, 6)$
5	$(0, 0, 0, 0, 4, 1)$
5	$(0, 0, 0, 0, 5, 0)$
4	$(0, 0, 0, 0, 4, 0)$
4	$(0, 0, 0, 4, 0, 0)$
3	$(0, 0, 3, 0, 0, 0)$
3	$(0, 3, 0, 0, 0, 0)$
3	$(3, 0, 0, 0, 0, 0)$

□

As already observed, MDS can be reduced to MSC by imposing $\mathcal{U} = V$ and $\mathcal{S} = \{N[v] \mid v \in V\}$. The size of the MSC instance obtained is at most $2n$. By simply combining this reduction with algorithm msc one obtains:

Corollary 3.2. *There is a $\mathcal{O}^*(2^{0.305(2n)}) = \mathcal{O}^*(2^{0.610n})$ algorithm for* MDS.

3.1 An Exponential Space Algorithm

The time complexity of msc can be reduced at the cost of an exponential space complexity via the *memorization* technique by Robson [22]. The general idea is the following: The algorithm keeps the solutions of all the subproblems solved. If the same subproblem turns up more than once, the algorithm is not to run a second time, but the already computed result is looked up. Note that the corresponding data structure can be implemented in such a way that the *query time* is logarithmic in the number of solutions stored [22].

Theorem 3.3. *Algorithm* msc*, modified as above, solves* MSC *in $\mathcal{O}^*(2^{0.299(|\mathcal{S}|+|\mathcal{U}|)})$ time.*

Proof. Let $k(\mathcal{S})$ be the size of the problem $\mathcal{S}$ considered, as defined in Theorem 3.1:

$$k(\mathcal{S}) = \sum_{i \geq 1} w_i\, n_i + \sum_{j \geq 1} v_j\, m_j,$$

where $w_i, v_j \in (0, 1]$. We recall that K is the set of the possible sizes of a subproblem $\mathcal{S}'$.

By $N_h(k)$ we denote the number of subproblems of size $h \in K$ solved by the algorithm to solve a problem of size k. From the proof of Theorem 3.1, $P_h(k) \leq \alpha^{k-h} \leq \alpha^{|\mathcal{S}|+|\mathcal{U}|-h}$, for some α depending on the values of the w_i and of the v_j. Each subproblem $\mathcal{S}'$ is obtained by removing some sets from $\mathcal{S}$ and some elements from $\mathcal{U}$. In other words:

$$\mathcal{S}' = del(\mathcal{U}^*, \mathcal{S} \backslash \mathcal{S}^*),$$

for some $\mathcal{S}^* \subseteq \mathcal{S}$ and $\mathcal{U}^* \subseteq \mathcal{U}$. Let $K(h)$ be the set of values $k' = |\mathcal{S}'| + |\mathcal{U}(\mathcal{S}')|$, where $\mathcal{S}'$ is a subproblem of $\mathcal{S}$ of size h. Since each subproblem is solved at most once, we have

$$P_h(k) \leq \sum_{k' \in K(h)} \binom{|\mathcal{S}| + |\mathcal{U}|}{k'}.$$

Specifically, for

$$k(h) = \min\{\max\{K(h)\},\, (|\mathcal{S}| + |\mathcal{U}|)/2\},$$

one has

$$\sum_{k' \in K(h)} \binom{|\mathcal{S}| + |\mathcal{U}|}{k'} \leq 2\binom{|\mathcal{S}| + |\mathcal{U}|}{k(h)}.$$

Let h' be the largest integer such that

$$2\binom{|\mathcal{S}| + |\mathcal{U}|}{k(h')} < \alpha^{|\mathcal{S}|+|\mathcal{U}|-h'}.$$

The total number $P(k)$ of subproblems solved satisfies:

$$P(k) \leq \sum_{h \in K, h \leq h'} 2\binom{|\mathcal{S}| + |\mathcal{U}|}{k(h')} + \sum_{h \in K, h > h'} \alpha^{|\mathcal{S}|+|\mathcal{U}|-h} \leq 2|K|\alpha^{|\mathcal{S}|+|\mathcal{U}|-h'}$$
$$= \mathcal{O}^*(|K|\alpha^{|\mathcal{S}|+|\mathcal{U}|-h'}).$$

Suppose for simplicity that $k(h') \leq (|\mathcal{S}| + |\mathcal{U}|)/2$. Note that, under this assumption,

$$k(h') = \max\{K(h')\} \leq \frac{h'}{\min_i\{w_i,\, v_i\}}.$$

Using Stirling's approximation one obtains

$$h' \geq \beta \min_i\{w_i,\, v_i\}(|\mathcal{S}| + |\mathcal{U}|),$$

where

$$\alpha^{1-\beta\ \min_i\{w_i,\, v_i\}} = \frac{1}{\beta^\beta(1-\beta)^{1-\beta}}.$$

By imposing the same values of w_i and of v_j as in Theorem 3.1, one obtains $\alpha = 1.2352\ldots < 1.2353$, $\min_i\{w_i, v_i\} = w_2 = 0.3774$, and $\beta > 0.0529$. Thus

$$P(k) = \mathcal{O}^*(|K|\alpha^{(1-\beta\, w_2)(|\mathcal{S}|+|\mathcal{U}|)}) = \mathcal{O}^*(1.2301^{|\mathcal{S}|+|\mathcal{U}|}) = \mathcal{O}^*(2^{0.299(|\mathcal{S}|+|\mathcal{U}|)}).$$

The cost of each query to the database is polynomial in k. Thus the cost of solving a problem of size $h \leq k$, excluding the cost of solving the corresponding subproblems (if any), is upper bounded by a polynomial $poly(k)$ of k. It follows that the time complexity of the algorithm is

$$\mathcal{O}^*(poly(k)P(k)) = \mathcal{O}^*(2^{0.299(|\mathcal{S}|+|\mathcal{U}|)}).$$

□

Corollary 3.4. *There is an algorithm which solves* `MDS` *in time* $\mathcal{O}^*(2^{0.299(2n)}) = \mathcal{O}^*(2^{0.598n})$.

4 An exponential lower bound

By carefully measuring the size of the subproblems, we obtained a much tighter running time bound. However the bound achieved might still be only a pessimistic estimation of the worst case running time of the algorithm. Therefore it is natural to ask for lower bounds: A lower bound may give an idea of how far is the bound computed from the real worst case running time.

Let us consider the $\mathcal{O}^*(2^{0.610n})$ polynomial-space `MDS` algorithm `mds` based on the reduction to `MSC` and (the polynomial-space version of) algorithm `msc`.

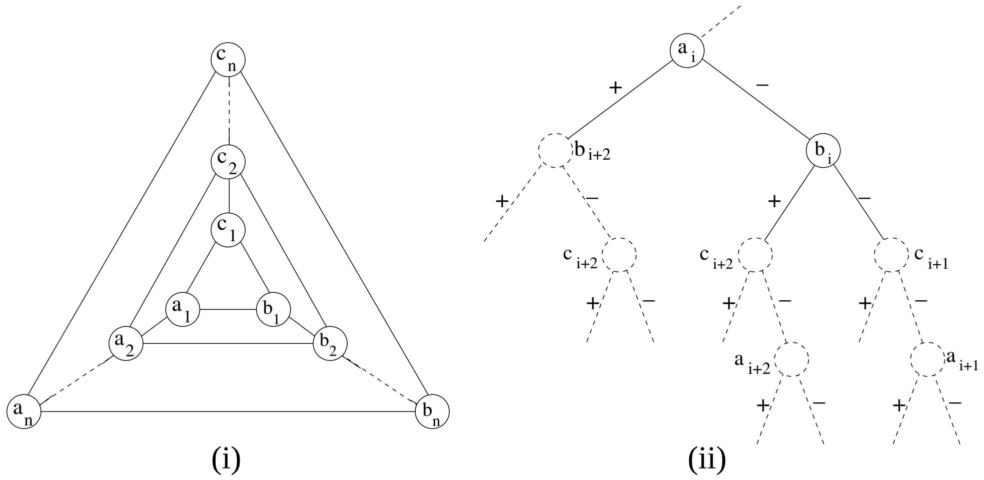

Figure 2: (i) graph G_n; (ii) a part of the search tree

Theorem 4.1. *The worst case running time of* `mds` *is* $\Omega(2^{n/3}) = \Omega(2^{0.333n})$.

Proof. Consider the following input graph G_n ($n \geq 1$): the node set of G_n is $\{a_i, b_i, c_i : 1 \leq i \leq n\}$. The edge set of G_n consists of two types of edges: for each $i = 1, 2 \ldots, n$, the vertices a_i, b_i and c_i induce a triangle T_i; and for each $i = 1, 2, ..., n-1$: $\{a_i, a_{i+1}\}$, $\{b_i, b_{i+1}\}$ and $\{c_i, c_{i+1}\}$ are edges. See Fig. 2.

Each node of the search tree corresponds to a subproblem of the `MSC` problem with input $(\mathcal{U}; \mathcal{S} = \{S_v : v \in V\})$ where $S_v = N[v]$.

We give a selection rule for the choice of the vertices v (respectively sets S_v) to be chosen for the branching. Clearly the goal is to choose them such that the number of nodes in the search tree obtained by the execution of algorithm `msc` on graph G_n is as large as possible.

In each round i, $i \in \{2, 3, \ldots, n-1\}$, we start with a pair $P = \{x_i, y_i\}$ of nodes (belonging to triangle T_i), where $\{x, y\} \subset \{a, b, c\}$. Initially $P = \{a_2, b_2\}$. Our choice makes sure that for each branching node x the cardinality of its set S_x is five in the current subproblem $\mathcal{S}$, and that no other rules of the algorithm will apply to a branching node than the one of line 5. Consequently, by line 7 of `msc` either the set S_v is taken into the set cover ($\mathcal{S} := del(\mathcal{S}, S_v)$), or S_v is removed ($\mathcal{S} := \mathcal{S} \setminus S_v$).

For each pair $P = \{x_i, y_i\}$ of nodes we branch in the following 3 ways
1) take S_{x_i}
2) remove S_{x_i}, and then take S_{y_i}
3) remove S_{x_i}, and then remove S_{y_i}

The following new pairs of nodes correspond to each of the three branches:
1) $P_1 = \{a_{i+2}, b_{i+2}, c_{i+2}\} \setminus x_{i+2}$
2) $P_2 = \{a_{i+2}, b_{i+2}, c_{i+2}\} \setminus y_{i+2}$
3) $P_3 = \{x_{i+1}, y_{i+1}\}$

On each pair P_j we recursively repeat the process. Thus of the three branches of T_i two are proceeded on T_{i+2} and one is proceeded on T_{i+1}.

Let $T(k)$ be the number of leaves in the search tree when all triangles up to T_k have been used for branching. Thus $T(k) = 2 \cdot T(k-2) + T(k-1)$, and hence $T(k) \geq 2^{k-2}$. Consequently the worst case number of leaves in the search tree of `msc` for a graph on n vertices is at least $2^{n/3-2}$. □

The lower bound above can be easily improved by considering disconnected graphs formed by several (disconnected) copies of a carefully chosen small subgraph.

We did not consider such lower bounds, since algorithm `mds` can be easily modified in order to invalidate them (it is sufficient to solve each disconnected subproblem separately, and then combine the partial solutions).

We may notice that there is a large gap between the $\mathcal{O}^*(2^{0.610n})$ upper bound and the $\Omega(2^{0.333n})$ lower bound. This could suggest the possibility that the analysis of algorithm `mds` can be further refined (possibly via a further refined measure of the size of the `MSC` instances).

5 Conclusions

We investigated the impact of different measures of the size of the problem in the analysis of exponential-time recursive algorithms. In particular, we considered the minimum dominating set problem. We showed how a more careful choice of the measure leads to a much tighter running time bound on the fastest know algorithm for the problem. Specifically, we reduced the time bound from $\mathcal{O}^*(2^{0.850n})$ to $\mathcal{O}^*(2^{0.598n})$ (without modifying the algorithm).

The impressive reduction of the running time achieved for minimum dominating set, suggests the possibility that the time complexity of many other exponential-time exact algorithms is largely overestimated because of a bad choice of the measure. Indeed, this could be the case also for our refined analysis of minimum dominating set. This possibility is somehow supported by the large gap between the $\mathcal{O}^*(2^{0.598n})$ upper bound and the $\Omega(2^{0.333n})$ lower bound we managed to prove.

Another natural problem to play with measure is Independent Set. The best running time $\mathcal{O}^*(2^{n/4})$ for this problem was claimed by Robson [23]. Though Robson's algorithm is extremely technical and complicated, the measure used in its analysis is very simple (the number of nodes in the graph). Can we refine the analysis of this algorithm via a different choice of the measure? Moreover, how fast really are simple algorithms for Independent Set?

References

[1] M. Alekhnovich, E.A. Hirsch, and D. Itsykon. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. Proceedings of the *31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, Springer LNCS vol. 3142, 2004, pp. 84–96.

[2] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3446^n)$: a no-MIS algorithm. Proceedings of the *36th IEEE Symposium on Foundations of Computer Science (FOCS 1995)*, pp. 444–452.

[3] J. M. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32:547–556, 2004.

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

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

[6] D. Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. Proceedings of the *12th ACM-SIAM Symposium on Discrete Algorithms (SODA 2001)*, pp. 329–337.

[7] D. Eppstein. Quasiconvex analysis of backtracking algorithms. Proceedings of the *15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pp. 781–790.

[8] F. V. Fomin, D. Kratsch, and I. Todinca. Exact algorithms for treewidth and minimum fill-in. Proceedings of the *31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, Springer LNCS vol. 3142, 2004, pp. 568–580.

[9] F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) algorithms for the dominating set problem. Proceedings of the 30th Workshop on Graph Theoretic Concepts in Computer Science (WG 2004), Springer LNCS vol. 3353, 2004, pp. 245-256.

[10] F. Grandoni. *Exact Algorithms for Hard Graph Problems*. PhD thesis, Università di Roma "Tor Vergata", Roma, Italy, Mar. 2004.

[11] F. Grandoni. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms*. To appear.

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

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

[14] K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. Proceedings of the *15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, p.328.

[15] T. Jian. An $O(2^{0.304n})$ algorithm for solving maximum independent set problem. *IEEE Transactions on Computers*, 35(9):847–851, 1986.

[16] E.L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters* 5(3):66–67, 1976.

[17] R. Niedermeier and P. Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, 36(1):63–88, 2000.

[18] P. Pudlak and R. Impaglazzio. A lower bound for DLL algorithms for k-SAT. A lower bound for DLL algorithms for k-SAT (preliminary version). Proceedings of the *11th ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pp. 128-136

[19] R. Paturi, P. Pudlak, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. Proceedings of the *39th IEEE Symposium on Foundations of Computer Science (FOCS 1998)*, pp. 628–637.

[20] B. Randerath and I. Schiermeyer. Exact algorithms for MINIMUM DOMINATING SET. Technical Report, zaik-469, Zentrum für Angewandte Informatik Köln, April 2004.

[21] B. Reed. Paths, stars and the number three. *Combinatorics, Probability and Computing* 5:277–295, 1996.

[22] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986.

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

[24] U. Schoning. A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems. Proceedings of the *40th IEEE Symposium on Foundations of Computer Science (FOCS 1999)*, pp. 410-414.

[25] R. Tarjan and A. Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, 6(3):537–546, 1977.

[26] R. Williams. A new algorithm for optimal constraint satisfaction and its implications. Proceedings of the *31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, Springer LNCS vol. 3142, 2004, pp. 1227–1237.

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