A Measure & Conquer Approach for the Analysis of Exact Algorithms*

Fedor V. Fomin† Fabrizio Grandoni‡ Dieter Kratsch§

February 20, 2009

Contents

*Preliminary parts of this article appeared in Proceedings of the *32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, Springer LNCS vol. 3580, 2005, pp. 191–203 and in Proceedings of the *17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, ACM, New York, 2006, pp. 18–25.

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway, fomin@ii.uib.no. Supported by the Norwegian Research Council.

‡Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, via del Politecnico 1, 00133 Roma, Italy, grandoni@disp.uniroma2.it.

§LITA, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France, kratsch@univ-metz.fr

Abstract

For more than 40 years Branch & Reduce exponential-time backtracking algorithms have been among the most common tools used for finding exact solutions of NP-hard problems. Despite that, the way to analyze such recursive algorithms is still far from producing tight worst-case running time bounds. Motivated by this we use an approach, that we call "Measure & Conquer", as an attempt to step beyond such limitations. The approach is based on the careful design of a non-standard measure of the subproblem size; this measure is then used to lower bound the progress made by the algorithm at each branching step. The idea is that a smarter measure may capture behaviors of the algorithm that a standard measure might not be able to exploit, and hence lead to a significantly better worst-case time analysis.

In order to show the potentialities of Measure & Conquer, we consider two well-studied NP-hard problems: minimum dominating set and maximum independent set. For the first problem, we consider the current best algorithm, and prove (thanks to a better measure) a much tighter running time bound for it. For the second problem, we describe a new, simple algorithm, and show that its running time is competitive with the current best time bounds, achieved with far more complicated algorithms (and standard analysis).

Our examples show that a good choice of the measure, made in the very first stages of exact algorithms design, can have a tremendous impact on the running time bounds achievable.

Keywords: Exact algorithm, dominating set, independent set.

1 Introduction

The aim of *exact algorithms* is to exactly solve NP-hard problems in the smallest possible (exponential) worst-case running time. This field dates back to the sixties and seventies [35, 64], and it has started to attract a growing interest in the last two decades [3, 4, 5, 6, 7, 8, 10, 27, 29, 36, 41, 42, 44, 45, 48, 49, 50, 51, 59, 61, 68]. There are several explanations to the increasing interest in exact algorithms:

- There are certain applications that require exact solutions of NP-hard problems, although this might only be possible for moderate input sizes.
- Approximation algorithms are not always satisfactory. Various problems are hard to approximate. For example, maximum independent set is hard to approximate within $O(n^{1-\varepsilon})$, for any constant $\varepsilon > 0$, unless $P = NP$ [70].
- A reduction of the base of the exponential running time, say from $O(2^n)$ to $O(2^{0.9\,n})$, increases the size of the instances solvable within a given amount of time by a constant *multiplicative* factor; running a given exponential algorithm on a faster computer can enlarge the mentioned size only by a (small) *additive* factor.
- The design and analysis of exact algorithms leads to a better understanding of NP-hard problems and initiates interesting new combinatorial and algorithmic challenges.

One of the major techniques in the design of exact algorithms is *Branch & Reduce*, which traces back to the paper of Davis and Putnam [12] (see also [11]). The basic idea is to apply a proper set of reduction rules, and then branch on two or more subproblems, which are solved recursively. The solutions to the subproblems are later used to derive a solution for the original problem. Branch & Reduce algorithms have been used for more than 40 years

to solve NP-hard problems. Despite that, the analytical tools available are still far from producing tight worst-case running time bounds for that kind of algorithm.

1.1 Measure & Conquer

Motivated by the limits of existing analytical tools for Branch & Reduce algorithms, we present here a new approach, that we call *Measure & Conquer*. To describe our method, and to show its potential, we apply it to the analysis of simple algorithms to solve two classical NP-hard problems: minimum dominating set and maximum independent set. In both cases we obtain considerably tighter time bounds with respect to the standard analysis.

The fastest known (Branch & Reduce) exact algorithms to solve NP-hard problems are often very complicated. Typically, they consist of a long list of non-trivial branching and reduction rules, and are designed by means of a long and tedious case distinction. However, their analysis is usually rather simple. A (standard) measure of the size of the subproblems is defined (e.g., number of vertices or edges of graphs, number of variables or clauses of CNF-formulas, etc.). This measure is used to lower bound the progress made by the algorithm at each branching step.

The idea behind Measure & Conquer is to focus on the choice of the measure. In fact, a more sophisticated measure may capture some phenomena which standard measures are not able to exploit, and hence lead to a tighter analysis of a *given* algorithm. We apply Measure & Conquer to the current best algorithm in [30, 31] for the minimum dominating set problem (`MDS`). The standard analysis of this algorithm given in [30, 31] provides an $O^*(2^{0.850\,n})$ bound on its running time[1]. By using a different measure, we are able to show that the *same* algorithm has in fact running time $O^*(2^{0.598\,n})$. We also consider the maximum independent set problem (`MIS`). For this problem, we present and analyze a very simple polynomial-space algorithm. Our algorithm, according to the standard analysis, performs very poorly: its running time is $O^*(2^{0.406\,n})$, which is much worse than the first non-trivial $O^*(2^{0.334\,n})$ algorithm by Tarjan and Trojanowski [64] for the same problem. However, thanks to a smarter measure, we manage to obtain an impressive refinement of the time analysis: the new time bound obtained (for the same algorithm) is $O^*(2^{0.287\,n})$. For a comparison, the current best results, which are obtained with far more complicated algorithms [59], are $O^*(2^{0.296\,n})$ in polynomial space and $O^*(2^{0.276\,n})$ in exponential space.

The results above show that a good choice of the measure can have a tremendous impact on the time bounds achievable, comparable to the impact of improved branching and reduction rules. Hence, finding a good measure should be at first concern when designing Branch & Reduce algorithms.

Despite the big improvements in the running time bounds, it might be that our refined analysis is still far from being tight. Hence, it is natural to ask for (exponential) 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. We prove $\Omega(2^{0.396\,n})$ and $\Omega(2^{0.142\,n})$ lower bounds on the worst-case time complexity of our `MDS` and `MIS` algorithms, respectively. The large gap between the upper and lower bounds for both algorithms suggests

[1]Throughout this paper we use a modified big-Oh notation that suppresses all polynomially bounded factors. For functions f and g we write $f(n) = O^*(g(n))$ if $f(n) = O(g(n)poly(n))$, where $poly(n)$ is a polynomial. Also while speaking about graph problems, we use n to denote the number of vertices in a graph.

the possibility that their analysis can be further refined (possibly by measuring the size of the subproblems in a further refined way).

1.2 Previous results

Non-standard measures. The idea of using non-standard measures is not new, though in most cases its real potential is not fully exploited. The most remarkable example is probably the seminal work by Eppstein et al. In a paper on 3-coloring and related problems [3], Beigel and Eppstein consider a reduction to constraint satisfaction, and measure the size of the constraint satisfaction problem with a linear combination of the number of variables with three and four values in their domain, respectively. A more sophisticated measure is introduced by Eppstein in the context of cubic-TSP [16]: let F be a given set of *forced* edges, that is edges that we assume belong to the optimum solution. For an input cubic graph $G = (V, E)$, the author measures the size of the problem in terms of $|V| - |F| - |C|$, where C is the set of 4-cycles which form connected components of $G - F$. Eppstein [17] also provides a general tool to analyze systems of multi-variate recurrences arising from the analysis of Branch & Reduce algorithms. He essentially shows that, from an asymptotic point of view and modulo polynomial factors, every set of multi-variate linear recurrences can be transformed into an equivalent set of univariate recurrences in terms of a proper linear combination of the original variables. Moreover, the coefficients of the linear combination can be found by solving a quasi-convex optimization problem. The last result is extensively used in this paper.

Minimum dominating set. MDS is a well known and well studied 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 [34] by Haynes, Hedetniemi, and Slater. The problem is hard to approximate: Unless $P = NP$ there is no polynomial time algorithm approximating MDS within a factor $c \log n$ for n-vertex graphs for some constant $c > 0$ [56]. The dominating set problem is also one of the basic problems in parameterized complexity [13]; it is W[2]-complete and thus it is unlikely that the problem is fixed parameter tractable. What are the best worst-case time complexities for MDS in n-vertex graphs $G = (V, E)$ that we can possibly hope for? It has been observed [25, 39] that there is no subexponential time (i.e. of running time $c^{o(n)}$ for some constant $c > 1$) algorithm solving MDS unless the complexity classes SNP and SUBEXP satisfy SNP $\subseteq$ SUBEXP which is considered to be unlikely. 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 $O^*(2^{cn})$, for some small constant $c < 1$. Although MDS is a natural and 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 independently published algorithms breaking the trivial "2^n-barrier". The algorithm of Fomin et al. [25] uses a deep graph-theoretic result due to Reed [57], 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 vertices of cardinality at most $3n/8$, thus the overall running time is $O^*(2^{0.955n})$. The algorithm of Randerath and Schiermeyer [53] uses combinatorial ideas (including matching techniques) to restrict the search space. The most time consuming part of their algorithm enumerates all subsets of vertices of cardinality at most $n/3$, thus the overall running time is $O^*(2^{0.919n})$. Finally, the fastest algorithm known prior to our work is due to Grandoni [30, 31], who described an $O^*(2^{0.850n})$ algorithm for

MDS. His algorithm is based on the standard reduction to minimum set cover, which will be adopted also in this paper.

Maximum independent set. MIS is one of the best studied NP-hard problems. By a recent result of Zuckerman [70] who succeeded in derandomizing the result of Håstad [33], no polynomial time approximation algorithm for MIS (unless $P = NP$) can provide an $O(n^{1-\varepsilon})$ guarantee for n-vertex graphs for any constant $\varepsilon > 0$. The problem is $W[1]$-complete [13], and thus it is probably not fixed parameter tractable. For reasons analogous to the case of MDS, it is also unlikely that MIS admits a sub-exponential time algorithm [39]. The design of exact algorithms for MIS has a long history. The first non-trivial exact algorithm solving MIS is due to Tarjan and Trojanowski (1977); it has running time $O^*(2^{0.334\,n})$ [64]. In 1986 Jian published an improved algorithm with running time $O^*(2^{0.304\,n})$ [42]. In the same year Robson provided an algorithm of running time $O^*(2^{0.296\,n})$ [59]. All these three algorithms are Branch & Reduce algorithms, and use polynomial space. In [59] Robson also showed how to speed up Branch & Reduce algorithms using a technique that is now called *Memorization*, and he established an $O^*(2^{0.276\,n})$ time algorithm that needs exponential space[2]. A significant amount of research was also devoted to solve the maximum independent set problem on sparse graphs [2, 8, 9, 27, 55].

Lower bounds. There are several results known on lower exponential bounds for different branching algorithms for SAT (see e.g. [1, 52]) but we are not aware of (non-trivial) lower bounds for existing exponential-time graph algorithms. One of the reasons to this could be that for most graph problems the construction of good lower bounds is often difficult even for very simple algorithms.

1.3 Related Work

The first papers with (non-trivial) exact algorithms appeared in the sixties and seventies. Classical examples are the $O^*(2^n)$ time algorithm for the travelling salesman problem with n cities by Held and Karp of 1962 [35] (see also the work of Kohn et al. [43]), the $O^*(2^{0.334\,n})$ time algorithm for the maximum independent set problem by Tarjan and Trojanowski of 1977 [64], Lawler's algorithm computing an optimal coloring of n-vertex graphs in time $O^*(2.4422^n)$ [49], and Horowitz-Sahni algorithm for the knapsack problem [38].

In the eighties the topic of exact algorithms was not in the mainstream of algorithmic research. However, the work of Monien and Speckenmeyer [50] on k-SAT, of Jian and Robson on the maximum independent set [42, 59], and of Schroeppel and Shamir on XSAT [63], prepared the ground for a rapid growing of the area which started in the late nineties.

It is impossible even to mention here all the results and problems studied for the last 10 years, so we give only a very short overview of the most important (from our point of view) recent results and techniques. A variety of results on k-SAT, and on 3-SAT in particular, improving on deterministic and probabilistic exact algorithms for the problem can be found in the literature. Among various techniques developed for k-SAT, let us mention the involved branching and reduction rules developed by Kullman [48], the randomized techniques of Paturi et al. [51], Schöning's approach based on random walks in the Boolean cube [61], and the

[2] In a technical report [60] Robson claims even better running times, both in polynomial and in exponential space. The description of his new algorithm, which is partially computer generated, takes almost 18 pages.

deterministic local search algorithm used by Dantsin et al. in [10]. See also [6, 36, 37, 41] for some other results in the area.

Many graph problems were studied from the viewpoint of exact algorithms. We already mentioned the work on maximum independent set and minimum dominating set together with the memorization technique. Another well studied problem is graph coloring [3, 7, 15], for which Bjorklund-Husfeldt and Koivisto recently obtained $O^*(2^n)$-time algorithms based on the inclusion-exclusion principle [4, 44]. There was also work done on treewidth [5, 24, 66], maximum cut [29, 45, 68], minimum feedback vertex set [18, 54] among many others. For more information, we refer to the surveys [21, 40, 62, 69].

Organization. The rest of this paper is organized as follows. In Section 2 we introduce some preliminary notions. In Section 3 and Section 4 we present our results on MDS and MIS, respectively. Conclusions are given in Section 5.

2 Preliminaries

Let $G = (V, E)$ be an n-vertex undirected, simple graph without loops. (For standard graph terminology, see e.g. [67]). Sometimes, we also use $V(G)$ for V and $E(G)$ for E. The (open) *neighborhood* of a vertex v is denoted by $N(v) = \{u \in V : uv \in E\}$, and its closed neighborhood by $N[v] = N(v) \cup \{v\}$. We let $d(v) = |N(v)|$ be the *degree* of v. By $N^d(v)$ we denote the set of vertices at distance d from v. In particular, $N^1(v) = N(v)$. Given a subset V' of vertices, $G[V']$ is the graph induced by V', and $G - V' = G[V \setminus V']$. Sometimes we will use $E(V')$ for $E(G[V'])$.

A set $D \subseteq V$ is called a *dominating set* for G if every vertex of G is either in D, or adjacent to some vertex 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 $\gamma(G)$. A problem closely related to MDS is the *minimum set cover* problem (MSC). In 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 $\cup_{R \in \mathcal{S}'} R = \mathcal{U}$. The *frequency* of $u \in \mathcal{U}$ is the number of subsets $R \in \mathcal{S}$ in which u is contained. We use $|u|$ to denote the frequency of u. For the sake of simplicity, we always assume in this paper that $\mathcal{S}$ covers $\mathcal{U}$, that is $\mathcal{U} = \mathcal{U}(\mathcal{S}) := \cup_{R \in \mathcal{S}} R$. With this assumption, an instance of MSC is univocally specified by $\mathcal{S}$. 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 vertices 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 $\{N[v] |\ v \in V\}$. In particular, every minimum set cover of $\{N[v] |\ v \in V\}$ corresponds to a minimum dominating set of G.

A set $I \subseteq V$ is called an *independent set* for G if the vertices of I are pairwise non adjacent. The *independence number* $\alpha(G)$ of a graph G is the maximum cardinality of an independent set of G. The *maximum independent set* problem (MIS) asks to determine $\alpha(G)$.

A set $A \subseteq E$ of edges of $G = (V, E)$ is an *edge cover*, if every vertex of G is incident to an edge of A; the edge set A is a *matching* if no vertex of G is incident to two edges of A.

Branch & Reduce algorithms. A typical Branch & Reduce algorithm for a given problem $\mathcal{P}$ works as follows. If $\mathcal{P}$ is a *base instance*, the problems is solved directly in polynomial

time. Otherwise the algorithm transforms the problem by applying a set of polynomial-time *reduction rules*. Then it branches, in polynomial-time, on two or more subproblems $\mathcal{P}_1, \ldots, \mathcal{P}_p$, according to a set of *branching rules*. Such subproblems are solved recursively, and the partial solutions obtained are eventually combined, in polynomial time, to get a solution for $\mathcal{P}$.

Branch & Reduce algorithms are usually analyzed in the following way. (For a more detailed description, see e.g. [48] and references therein). Suppose we wish to find a time bound in terms of a given measure k of the input size. Assume that the depth of the search tree is polynomially bounded (which is trivially true in most cases). It is sufficient to bound the maximum number $P(k)$ of base instances generated by the algorithm: the running time will be $O^*(P(k))$. If $\mathcal{P}$ is a base instance, trivially $P(k) = 1$. Otherwise, let $k_i = k - \Delta\, k_i < k$ be the size of subproblem $\mathcal{P}_i$ for a given branching. It follows that

$$P(k) \leq \sum_{i=1}^{p} P(k_i),$$

for every feasible combination of subproblems $\mathcal{P}_1, \ldots, \mathcal{P}_p$. It turns out that $P(k) \leq \lambda^k$, where $\lambda \geq 1$ is the largest root of the set of equations of the kind

$$1 = \sum_{i=1}^{p} x^{-\Delta\, k_i},$$

obtained by considering every feasible *branching vector* $\Delta = (\Delta\, k_1, \ldots, \Delta\, k_p)$. The root $r(\Delta)$ associated to a given branching vector Δ is sometimes called *branching factor*. For a given Δ, $r(\Delta)$ can be easily found numerically. We say that a branching vector Δ *dominates* a branching vector Δ' if $\Delta \leq \Delta'$, i.e. Δ is component-wise not larger than Δ'. It is not hard to see that, when $\Delta \leq \Delta'$, $r(\Delta) \geq r(\Delta')$. Hence, with respect to the running time analysis, it is sufficient to consider a dominating set of branching vectors. For a similar reason, each time we replace the branching vector of a feasible branching with a branching vector dominating it, we obtain a pessimistic estimate of the running time. These properties will be extensively used in this paper.

In the standard analysis, k is both the measure used in the analysis and the quantity in terms of which the final time bound is expressed. However, one is free to use any, possibly sophisticated, measure k' in the analysis, provided that $k' \leq f(k)$ for some known function f. This way, one achieves a time bound of the kind $O^*(\lambda^{k'}) = O^*(\lambda^{f(k)})$, which is in the desired form. As we will see, a proper choice of k' can lead to a *better balanced* set of recurrences, and hence to an improved running time bound.

3 The Minimum Dominating Set Problem

In [30, 31] Grandoni describes an $O^*(2^{0.930\, n})$ algorithm `mds` for `MDS` based on the following approach. He first reduces the input problem $G = (V, E)$ to an equivalent instance $(\mathcal{S}, \mathcal{U}) = (\{N[v] : v \in V\}, V)$ of `MSC`. Then he solves $(\mathcal{S}, \mathcal{U})$ via a simple `MSC` algorithm `msc`. Algorithm `msc` is described in Section 3.1[3]. It is shown that `msc` runs in time $O^*(2^{0.465(|\mathcal{S}|+|\mathcal{U}|)})$. As a

[3]In fact, for ease of presentation, we consider here a slightly modified version of `msc`, which has the same running time from the point of view of the standard analysis.

Figure 1 Algorithm `msc` for the minimum set cover problem.

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

consequence, the running time of `mds` is $O^*(2^{0.465(n+n)}) = O^*(2^{0.930\,n})$. In Section 3.2 we show, thanks to a refined measure, that in fact `msc` runs in time $O^*(2^{0.305(|\mathcal{S}|+|\mathcal{U}|)})$, and hence `mds` in time $O^*(2^{0.610\,n})$. This result is complemented by a $\Omega(2^{0.396\,n})$ lower bound on the running time of `mds` (see Section 3.3). Algorithm `mds` runs in polynomial space. Grandoni shows how to reduce the running time of `mds` to $O^*(2^{0.850\,n})$ using exponential space. According to our refined measure, the exponential-space running time bound can be refined to $O^*(2^{0.598\,n})$ (see Section 3.4).

3.1 The Algorithm

Before describing `msc`, we need some further preliminary notions. Recall that, without loss of generality, we assume $\mathcal{U} = \mathcal{U}(\mathcal{S}) := \cup_{R\in\mathcal{S}} R$. Hence, a set cover instance can be specified by providing $\mathcal{S}$ only. We observe that:

Lemma 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* $u \in \mathcal{U}(\mathcal{S})$ *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 of Lemma 1.

We also recall that `MSC` is solvable in polynomial time and space when all the subsets of $\mathcal{S}$ are of cardinality two, by applying the following standard reduction to maximum matching. Consider the graph $\widetilde{G}$ which has a vertex u for each $u \in \mathcal{U}$, and an edge uv for each subset $S = \{u, v\}$ in $\mathcal{S}$. Note that a minimum set cover for $\mathcal{S}$ corresponds to a minimum edge cover[4] of $\widetilde{G}$. To compute a minimum edge cover of $\widetilde{G}$, it is sufficient to compute a maximum matching M in $\widetilde{G}$. Then, for each unmatched vertex u, we add to M an arbitrary edge incident to u (if no such edge exists, there is no set cover at all). The final set M is the desired edge cover of $\widetilde{G}$ (and set cover of G). In the following, we will call `2-msc` the algorithm described above.

Algorithm `msc` is described in Figure 1. If $|\mathcal{S}| = 0$ (line 1), $\texttt{msc}(\mathcal{S}) = 0$. Otherwise, the algorithm tries to reduce the size of the problem without branching, by applying one of the Properties 1 and 2 of Lemma 1. Specifically, if there are two sets S and R, $S \subseteq R$, the algorithm returns (line 2)

$$\texttt{msc}(\mathcal{S}) = \texttt{msc}(\mathcal{S} \setminus \{S\}).$$

[4]An edge cover of a graph $G = (V, E)$ is a subset $E' \subseteq E$ of edges such that each vertex $v \in V$ is the endpoint of at least one edge $e \in E'$.

If there is an element u which is contained in a unique set S, the algorithm returns (line 3)

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

where

$$del(S, \mathcal{S}) = \{Z | Z = R \setminus S \neq \emptyset, R \in \mathcal{S}\}$$

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 neither of the two properties above applies, the algorithm takes a set $S \in \mathcal{S}$ of maximum cardinality (line 4). If $|S| = 2$, the algorithm directly solves the problem (in polynomial time and space) via 2-msc (line 5). Otherwise (line 6), it branches on the two subproblems $S_{IN} = del(S, \mathcal{S})$ (the case where S belongs to the minimum set cover) and $S_{OUT} = \mathcal{S} \setminus \{S\}$ (corresponding to the case S is not in the minimum set cover), and returns

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

Notice that with simple modifications, the algorithm can also provide one minimum set cover (besides its cardinality). In fact, at each recursive call of msc some set S is either implicitly included in (lines 3 and 6) or implicitly excluded from (lines 2 and 6) the minimum set cover under construction. In the first case, we say that S is *selected*, and otherwise it is *discarded*.

The standard analysis. To emphasize the importance of the choice of the measure, we sketch the analysis of the algorithm with a simple measure (taken from [31]). Let us choose the following measure $k = k(\mathcal{S})$ of the size of a MSC instance $\mathcal{S}$,

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

Let $P(k)$ be the maximum number of base instances generated by the algorithm to solve a problem of size k. If one of the conditions of lines 1 and 5 holds, the algorithm directly solves the problem, and hence $P(k) = 1$. If one of the conditions of lines 2 and 3 is satisfied, $P(k) \leq P(k-1)$ since at least one set is removed from the problem. Otherwise, let S be the set taken in line 4 ($|S| \geq 3$). The algorithm branches on the two subproblems $\mathcal{S}_{OUT} = \mathcal{S} \setminus \{S\}$ and $\mathcal{S}_{IN} = del(S, \mathcal{S})$. 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 $P(k) \leq P(k-1) + P(k-4)$. We conclude that $P(k) \leq \lambda^k$, where $\lambda = 1.3802\ldots < 1.3803$ 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 $P(k)$. Moreover, solving each subproblem takes polynomial time. Thus the complexity of the algorithm is $O^*(P(k)) = O^*(\lambda^k) = O^*(1.3803^{|\mathcal{S}|+|\mathcal{U}|}) = O^*(2^{0.465(|\mathcal{S}|+|\mathcal{U}|)})$.

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

3.2 The Analysis

In this section we present a refined analysis of msc, based on a more sophisticated measure of the size of the subproblems.

Our refined measure 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 off in the 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 in the long term, since sets of cardinality one can be filtered out. Both phenomena are not taken into account in the measure used in [31]. With that measure, by removing one set (element), we decrease the size of the problem by one, no matter what is the cardinality of the set (frequency of the element) considered.

This suggests the idea of giving 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 $|u| = j$. We will use the following measure $k = k(\mathcal{S})$ of the size of $\mathcal{S}$:

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

where the weights $\alpha_i, \beta_j \in (0, 1]$ will be fixed in the following. Note that our choice of the weights ensures that $k \leq |\mathcal{S}| + |\mathcal{U}|$. Thanks to this constraint, we will be able at the end of the analysis to provide a bound in the desired form $O^*(\lambda^{|\mathcal{S}|+|\mathcal{U}|})$.

In order to simplify the running time analysis, we will make the following extra assumptions:

(*a*) $0 < \alpha_i \leq \alpha_{i+1}$ and $0 < \beta_i \leq \beta_{i+1}$ for $i \geq 2$;

(*b*) $\alpha_1 = \beta_1 = 0$;

(*c*) $\alpha_i = \beta_i = 1$ for $i \geq 6$.

The first assumption reflects our intuition that instances with larger sets and with elements of larger frequency are harder to solve, and hence should have a larger size according to our measure. In view of that, the second assumption is clear: sets of cardinality one and elements of frequency one can be removed very "cheaply", i.e. without branching, and thus should not contribute to the size of the problem. The last assumption is simply due to the fact that we are not able to deal with an unbounded number of weights. We experimentally observed that further increasing the number of distinct weights does not improve the analysis significantly.

The quantities

$$\Delta \, \alpha_i = \alpha_i - \alpha_{i-1}, \; i \geq 2 \quad \text{and} \quad \Delta \, \beta_i = \beta_i - \beta_{i-1}, \; i \geq 2,$$

turn out to be useful in the analysis. Intuitively, $\Delta \, \alpha_i$ ($\Delta \, \beta_i$) 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 $i - 1$. We make one last simplifying assumption:

(*d*) $\Delta\alpha_i \geq \Delta\alpha_{i+1}$, for $i \geq 2$,

that is the α_i's are increasing at decreasing speed. This last assumption helps to simplify the analysis, and turns out to be non-restrictive.

Theorem 1 *Algorithm* `msc` *solves* `MSC` *in* $O^*(2^{0.305(|\mathcal{U}|+|\mathcal{S}|)})$ *time and polynomial space.*

Proof. The correctness of the algorithm is straightforward. Moreover, its space complexity is trivially polynomial.

Recall that $P(k)$ denotes the maximum number of base instances generated by the algorithm to solve a problem of size k. Clearly, $P(0) = 1$, since in this case the algorithm never branches. Consider the case $k > 0$ (which implies $\mathcal{S} \neq \emptyset$). If one of the conditions of lines 2 and 3 holds, one set S is removed from $\mathcal{S}$. Thus we get $P(k) \leq P(k - \alpha_{|S|})$, where $\alpha_{|S|} \geq 0$ by Assumptions (a) and (b).

Otherwise, let S be the subset selected in line 4. If $|S| = 2$, no subproblem is generated ($P(k) = 1$). Otherwise ($|S| \geq 3$), `msc` generates two subproblems $\mathcal{S}_{IN} = del(S, \mathcal{S})$ and $\mathcal{S}_{OUT} = \mathcal{S} \setminus \{S\}$.

We wish to lower bound the difference between the size of $\mathcal{S}$ and the size of the two subproblems $\mathcal{S}_{IN}$ and $\mathcal{S}_{OUT}$. Consider the subproblem $\mathcal{S}_{OUT}$. The size of $\mathcal{S}_{OUT}$ decreases by $\alpha_{|S|}$ 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 one. Hence

$$\sum_{i \geq 1} r_i = \sum_{i \geq 2} r_i = |S|.$$

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 $\mathcal{S}_{OUT}$ decreases by $\Delta\,\beta_i$. Thus the overall reduction of the size of $\mathcal{S}_{OUT}$ due to the reduction of the frequencies is at least

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

where we used the fact that $\Delta\,\beta_i = 0$ for $i \geq 7$ (Assumption (c)).

Suppose that $r_2 > 0$, and let $R_1, R_2, \ldots, R_h$, $1 \leq h \leq r_2$, be the sets of $\mathcal{S}$ distinct from S, which share at least one element of frequency two with S. When we discard S, we must select all the sets R_i before the next branching (on two subproblems). Suppose R_i, $1 \leq i \leq h$, shares $r_{2,i}$ elements of frequency two with S. Then $|R_i| \geq r_{2,i} + 1$, since otherwise we would have $R \subseteq S$, which is excluded by line 2. Thus, by Assumption (a), the reduction of the size of the problem due to the removal of R_i is $\alpha_{|R_i|} \geq \alpha_{r_{2,i}+1}$. Note that $r_{2,i} \leq |R_i| - 1 < |S|$, being S of maximum cardinality by assumption: this is used in the case analysis below. We also observe that, by selecting the R_i's, we remove at least one element $u \notin S$, thus gaining an extra $\beta_{|u|} \geq \beta_2$ (here we use Assumption (a) again). By a simple case analysis, which we present here in a slightly weakened form, the total reduction of the size of the problem due to the removal of the R_i's is at least

$$\Delta\,k'_{|S|,r_2} = \begin{cases} 0 & \text{if } r_2 = 0; \\ \beta_2 + \alpha_2 & \text{if } r_2 = 1; \\ \beta_2 + \min\{2\alpha_2, \alpha_3\} = \beta_2 + \alpha_3 & \text{if } r_2 = 2; \\ \beta_2 + \min\{3\alpha_2, \alpha_2 + \alpha_3\} = \beta_2 + \alpha_2 + \alpha_3 & \text{if } r_2 = 3, |S| = 3; \\ \beta_2 + \min\{3\alpha_2, \alpha_2 + \alpha_3, \alpha_4\} = \beta_2 + \alpha_4 & \text{if } r_2 \geq 3, |S| \geq 4. \end{cases}$$

Above we used the fact that, by Assumptions (b) and (d),

$$\min\{2\alpha_2, \alpha_3\} = \min\{\Delta\alpha_2 + \alpha_2, \Delta\alpha_3 + \alpha_2\} = \Delta\alpha_3 + \alpha_2 = \alpha_3,$$

and

$$\min\{\alpha_2+\alpha_3,\alpha_4\}=\min\{\Delta\alpha_2+\alpha_3,\Delta\alpha_4+\alpha_3\}=\Delta\alpha_4+\alpha_3=\alpha_4.$$

Consider now the subproblem $\mathcal{S}_{IN}$. The size of $\mathcal{S}_{IN}$ decreases by $\alpha_{|S|}$ because of the removal of S. Let $r_{\geq i}=\sum_{j\geq i} r_j$ be the number of elements of S of frequency at least i. Consider an element $u\in S$ of frequency i ($i\geq 2$). The size of $\mathcal{S}_{IN}$ further decreases by β_i because of the removal of u. Thus the overall reduction due to the removal of the elements u of S is

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

where we used the fact that $\beta_i=1$ for $i\geq 7$ (Assumption (c)). 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 $\Delta\,\alpha_{|R|}\geq\Delta\,\alpha_{|S|}$ (Assumption (d)). Thus the overall reduction of $\mathcal{S}_{IN}$ due to the reduction of the cardinalities of the sets R is at least:

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

Note that this quantity is 0 for $|S|\geq 7$.

Putting all together, for any tuple $t=(|S|,r_2,\ldots,r_6,r_{\geq 7})$ with $|S|\geq 3$ and $\sum_{i=2}^{6} r_i+r_{\geq 7}=|S|$, we obtain the following recurrence

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

where

- $\Delta\,k_{OUT}(t):=\alpha_{|S|}+\sum_{i=2}^{6} r_i\,\Delta\,\beta_i+\Delta\,k'_{|S|,r_2}$,
- $\Delta\,k_{IN}(t):=\alpha_{|S|}+\sum_{i=2}^{6} r_i\,\beta_i+r_{\geq 7}+\Delta\,\alpha_{|S|}\left(\sum_{i=2}^{6}(i-1)\,r_i+6\cdot r_{\geq 7}\right)$.

For every fixed 8-tuple $(\alpha_2,\alpha_3,\alpha_4,\alpha_5,\beta_2,\beta_3,\beta_4,\beta_5)$ the quantity $P(k)$ is upper bounded by λ^k, where λ is the largest root of the set of equations

$$1=x^{-\Delta\,k_{OUT}(t)}+x^{-\Delta\,k_{IN}(t)}$$

corresponding to different combinations of values of $|S|$ and of the r_i's. Thus the estimation of $P(k)$ boils down to choosing the weights minimizing λ. This optimization problem is interesting in its own, and we refer to Eppstein's work [17] on quasi-convex programming for a general treatment of this kind of problem.

We crucially observe that the bound on λ given by the recurrences with $|S|\geq 8$ is not larger than the bound on λ given by the recurrences with $|S|=7$ (the latter recurrences *dominate* the first ones). In fact, consider any tuple $t=(|S|,r_2,\ldots,r_6,r_{\geq 7})$ with $|S|\geq 8$ and $\sum_{i=2}^{6} r_i+r_{\geq 7}=|S|$. Let $t'=(|S'|,r'_2,\ldots,r'_6,r'_{\geq 7})$, with $|S'|=7$, $0\leq r'_i\leq r_i$, $0\leq r'_{\geq 7}\leq r_{\geq 7}$, and $\sum_{i=2}^{6} r'_i+r'_{\geq 7}=7$. Observe that t' is a feasible tuple for sets of cardinality 7. Moreover, $\alpha_{|S|}=\alpha_{|S'|}=1$, $\Delta\,\alpha_{|S|}=\Delta\,\alpha_{|S'|}=0$, and $\Delta\,k'_{|S|,r_2}\geq\Delta\,k'_{|S'|,r'_2}$. Hence $\Delta\,k_{OUT}(t)\geq\Delta\,k_{OUT}(t')$ and $\Delta\,k_{IN}(t)\geq\Delta\,k_{IN}(t')$, i.e. the branching vector $(\Delta\,k_{OUT}(t'),\Delta\,k_{IN}(t'))$ dominates the branching vector $(\Delta\,k_{OUT}(t),\Delta\,k_{IN}(t))$. Therefore we can restrict our attention to the case

Table 1 The worst-case recurrences for `msc`.

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

$3 \leq |S| \leq 7$. This way, we have to consider a large but finite number of recurrences only. (The actual number of recurrences is 1688).

To find the (nearly) optimal weights we used a computer program, which is based on the following randomized local search strategy. We start from a feasible choice of the weights W, and we compute the corresponding value $\lambda = \lambda(W)$. Then we randomly perturb the weights: if the new weights W' obtained are feasible, and $\lambda(W') < \lambda(W)$, we set W equal to W'. The perturbation of the weights is performed in the following way. For a proper value $\delta > 0$ and for each weight w' independently, we add to w' a random quantity sampled uniformly at random in the interval $[-\delta, \delta]$. The value of δ is reduced if no improvement of $\lambda(W)$ is obtained for a long number of steps. The process halts when the value of δ drops below a fixed (small) threshold. Our algorithm turns out to be very fast and sufficiently accurate in practice even for a large number of weights and recurrences. The outcome of the program in the case considered was:

$$\alpha_i = \begin{cases} 0.377443 & \text{if } i = 2, \\ 0.754886 & \text{if } i = 3, \\ 0.909444 & \text{if } i = 4, \\ 0.976388 & \text{if } i = 5, \end{cases} \quad \text{and} \quad \beta_i = \begin{cases} 0.399418 & \text{if } i = 2, \\ 0.767579 & \text{if } i = 3, \\ 0.929850 & \text{if } i = 4, \\ 0.985614 & \text{if } i = 5, \end{cases}$$

which yields $\lambda \leq 1.2352\ldots < 2^{0.305}$. In Table 1 the values of $|S|$ and of the r_i's of the worst-case recurrences are listed.

Now we observe that at each branching step we remove at least one set. Moreover, the time spent for each branching is upper bounded by a polynomial in $|\mathcal{S}| + |\mathcal{U}|$. Therefore, the overall running time of the algorithm is

$$O^*(P(k)) = O^*(\lambda^k) = O^*(2^{0.305(|\mathcal{U}|+|\mathcal{S}|)}).$$

We remark that, for any feasible choice of the weights, the corresponding value of λ gives a feasible upper bound on the running time of the algorithm (though possibly not the best possible). Moreover, in order to check that a given λ is feasible for given weights, it is sufficient to check that $\lambda^k \geq \lambda^{k-\Delta k_{OUT}(t)} + \lambda^{k-\Delta k_{IN}(t)}$ (i.e., $1 \geq \lambda^{-\Delta k_{OUT}(t)} + \lambda^{-\Delta k_{IN}(t)}$) for all the feasible tuples t. In the Appendix (Figure 9) we provide the pseudo-code of a program which can be used to check the condition above (and hence the correctness of the claim). □

Recall that `mds` is the `MDS` algorithm based on the standard reduction to `MSC` and on `msc`.

Corollary 1 *Algorithm* `mds` *solves* `MDS` *in time* $O^*(2^{0.305(2n)}) = O^*(2^{0.610n})$ *and polynomial space.*

Proof. The claim follows trivially from Theorem 1, observing that the size of the MSC instance $(\mathcal{S},\mathcal{U})$ obtained satisfies $|\mathcal{S}| = |\mathcal{U}| = n$. □

Remark 1 *The analysis of* msc *can be slightly improved by imposing* $\beta_i = 0.98232$ *for* $i \geq 6$ *(instead of* $\beta_i = 1$*). This way, it is possible to show that* MSC *is solvable in time* $O^*(1.23728^{|\mathcal{S}|+0.98232|\mathcal{U}|})$*. As a consequence,* MDS *can be solved in time* $O^*(1.23728^{1.98232\,n}) = O^*(2^{0.609\,n})$*. Since the improvement obtained is small, we do not give the details of the refined analysis here.*

3.3 An Exponential Lower Bound

By carefully measuring the size of the subproblems, we obtained a much tighter running time bound for mds. 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 an (exponential) lower bound on the running time of the algorithm, which may give an idea of how far is our analysis from being tight.

Theorem 2 *The worst-case running time of* mds *is* $\Omega(3^{n/4}) = \Omega(2^{0.396\,n})$.

Proof. Consider a graph $G_\ell = (V, E)$ consisting of $\ell \geq 1$ disconnected copies of a cycle of length 4. Let $\mathcal{P}_\ell = (\mathcal{S},\mathcal{U}) = (\{N[v] : v \in V\}, V)$ be the MSC instance associated to G_ℓ. We will show that msc can branch two times consecutively on sets related to a given cycle, generating 3 instances of $\mathcal{P}_{\ell-1}$. This implies by an easy induction that the overall number of subproblems generated, and hence the running time of the algorithm, is lower bounded by $3^\ell = 3^{n/4}$.

Consider any given cycle a, b, c, d. We will denote by S_v the set associated to vertex v (initially, $S_v = N[v]$). Note that set $S_a = \{a, b, d\}$ has the largest cardinality in the problem, i.e. 3. Moreover, the conditions of lines 2 and 3 do not apply. Hence msc can branch at line 6 on set S_a. Consider the subproblem where S_a is selected: sets $S_b = \{c\}$, $S_c = \{c\}$ and $S_d = \{c\}$ are either discarded at line 2 or selected at line 3 (without branching); the resulting subproblem is of type $\mathcal{P}_{\ell-1}$. Consider now the subproblem where S_a is discarded. In this case the conditions of lines 2 and 3 do not apply, and $S_b = \{a, b, c\}$ is a set of maximum cardinality 3: therefore msc can branch on S_b at line 6. By the same argument as before, in the subproblem where S_b is selected (and S_a discarded) the application of lines 2 and 3 gives a subproblem of type $\mathcal{P}_{\ell-1}$. On the other hand, in the subproblem where S_b is discarded (together with S_a), sets $S_c = \{b, c, d\}$ and $S_d = \{a, c, d\}$ are both selected at line 3 (they are the unique sets covering b and a, respectively): also in this case we obtain a subproblem of type $\mathcal{P}_{\ell-1}$. The claim follows. □

3.4 An Exponential Space Algorithm

The time complexity of msc, and hence of mds, can be reduced at the cost of an exponential space complexity via the *memorization* technique by Robson [59]. 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 [59]. Every subproblem can be encoded via a subset of $(\mathcal{S},\mathcal{U})$. Hence the number of distinct subproblems is upper bounded by $2^{|\mathcal{S}|+|\mathcal{U}|}$, which implies that the query time is polynomial in $|\mathcal{S}| + |\mathcal{U}|$.

Here we consider a simple variant of the technique above, where we do not store all the solutions computed, but only the ones corresponding to subproblems where line 6 applies (that is, the subproblems which branch on two further subproblems). This is not crucial, but it helps to simplify the analysis.

Theorem 3 *Algorithm* `msc`*, modified as above, solves* `MSC` *in* $O^*(2^{0.299(|\mathcal{S}|+|\mathcal{U}|)})$ *time and exponential space.*

Proof. Consider the set $\mathcal{P}$ of those subproblems generated during the execution of the algorithm on which the algorithm branches at line 6. In particular, none of these subproblems contains a set of cardinality one nor an element of frequency one. Let $P_h(k)$ be the maximum number of such subproblems of size h, $0 \leq h \leq k$. By basically the same analysis as in Theorem 1, $P_h(k) \leq 1.2353^{k-h} \leq 1.2353^{k'-h}$, where $k' := |\mathcal{S}| + |\mathcal{U}|$.

Consider one such subproblem of size h. Observe that it can be encoded via a pair $(\mathcal{S}',\mathcal{U}')$, where $\mathcal{S}' \subseteq \mathcal{S}$ and $\mathcal{U}' \subseteq \mathcal{U}$. Since the problem considered does not contain any set of cardinality one nor any element of frequency one, we have that

$$|\mathcal{S}'| + |\mathcal{U}'| \leq \lfloor h/\min\{\alpha_2, \beta_2\}\rfloor = \lfloor h/0.377443\rfloor =: h'.$$

As a consequence, since no subproblem is solved more than once, $P_h(k)$ is also upper bounded by

$$P_h(k) \leq \sum_{i\leq h'} \binom{k'}{i}.$$

Observe that, the number of different weights being a constant, the number of possible distinct feasible values of h is a polynomial in k. Putting things together,

$$\begin{aligned}
|\mathcal{P}| &\leq \sum_h \min\left\{1.2353^{k'-h}, \sum_{i\leq h'}\binom{k'}{i}\right\} \\
&= O^*(\sum_{h'>k'/2} 1.2353^{k'-h'\min\{\alpha_2,\beta_2\}} + \sum_{h'\leq k'/2} \min\left\{1.2353^{k'-h'\min\{\alpha_2,\beta_2\}}, \binom{k'}{h'}\right\}) \\
&= O^*(2^{0.248\,k'} + \max_{h'\leq k'/2} \min\left\{1.2353^{k'-h'\min\{\alpha_2,\beta_2\}}, \binom{k'}{h'}\right\}).
\end{aligned}$$

Applying Stirling's formula,

$$\max_{h'\leq k'/2} \min\left\{1.2353^{k'-h'\min\{\alpha_2,\beta_2\}}, \binom{k'}{h'}\right\} = O^*(1.2353^{k'-0.01996k'}) = O^*(2^{0.299\,k'}).$$

Hence, $|\mathcal{P}| = O^*(2^{0.299\,k'})$. At each branching step the algorithm removes at least one set. Thus the total number of subproblems is $O^*(|\mathcal{P}|)$. Moreover, the cost of each query to the database is polynomial in k. It follows that the running time of the algorithm is $O^*(2^{0.299\,k'}) = O^*(2^{0.299(|\mathcal{S}|+|\mathcal{U}|)})$. □

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

4 The Maximum Independent Set Problem

In this section we present our maximum independent set algorithm `mis`. Our algorithm branches by imposing that some vertices belong or do not belong to the maximum independent set to be computed: we call the vertices of the first kind *selected*, and the other ones *discarded*.

4.1 Folding and Mirroring

Before presenting `mis`, we describe some simple properties of maximum independent sets. Recall that $\alpha(G)$ denotes the size of a maximum independent set of a graph G. First of all, we observe that:

Lemma 2 *Let G be a graph.*

- **(connected components)** *For every connected component C of G,*

$$\alpha(G) = \alpha(C) + \alpha(G - C).$$

- **(dominance)** *If there are two vertices v and w such that $N[w] \subseteq N[v]$ (w* dominates *v), then*

$$\alpha(G) = \alpha(G - \{v\}).$$

We will use the following folding operation, which is a special case of the *struction* operation defined in [14], and which was introduced in the context of exact algorithm for `MIS` in [2, 8]. A vertex v is *foldable* if $N(v) = \{u_1, u_2, \ldots, u_{d(v)}\}$ contains no anti-triangle[5]. *Folding* a given foldable vertex v of G is the process of transforming G into a new graph $\widetilde{G}(v)$ by:

(1) adding a new vertex u_{ij} for each anti-edge u_iu_j in $N(v)$;

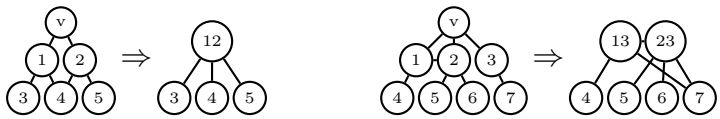
(2) adding edges between each u_{ij} and the vertices in $N(u_i) \cup N(u_j)$;

(3) adding one edge between each pair of new vertices;

(4) removing $N[v]$.

Note that vertices of degree at most two are always foldable. Examples of folding are given in Figure 2. The following simple property holds.

Figure 2 Folding of a vertex v.



Lemma 3 (folding) *Consider a graph G, and let $\widetilde{G}(v)$ be the graph obtained by folding a foldable vertex v. Then*

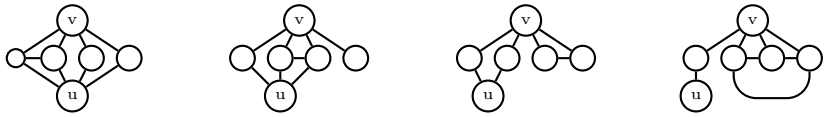
$$\alpha(G) = 1 + \alpha(\widetilde{G}(v)).$$

[5] An anti-triangle is a triple of vertices which are pairwise not adjacent. Similarly, an anti-edge is a pair of non-adjacent vertices.

Proof. Let S be a maximum independent set of G. If $v \in S$, then $S \setminus \{v\}$ is an independent set of $\widetilde{G}(v)$. Otherwise, S contains at least one vertex of $N(v)$ (since it is of maximum cardinality). If $N(v) \cap S = \{u\}$, then $S \setminus \{u\}$ is an independent set of $\widetilde{G}(v)$. Otherwise, it must be $N(v) \cap S = \{u_i, u_j\}$, for two non-adjacent vertices u_i and u_j (since $N(v)$ does not contain any anti-triangle by assumption). In this case $S \cup \{u_{ij}\} \setminus \{u_i, u_j\}$ is an independent set of $\widetilde{G}(v)$. It follows that $\alpha(G) \leq 1 + \alpha(\widetilde{G}(v))$. A similar argument shows that $\alpha(G) \geq 1 + \alpha(\widetilde{G}(v))$. □

We eventually introduce the following useful notion of mirror. Given a vertex v, a *mirror* of v is a vertex $u \in N^2(v)$ such that $N(v) \setminus N(u)$ is a (possibly empty) clique. We denote by $M(v)$ the set of mirrors of v. Examples of mirrors are given in Figure 3. Intuitively, when

Figure 3 Example of mirrors: u is a mirror of v.



we discard a vertex v, we can discard its mirrors as well without modifying the maximum independent set size. This intuition is formalized in the following lemma.

Lemma 4 (mirroring) *For any graph G and for any vertex v of G,*

$$\alpha(G) = \max\{\alpha(G - \{v\} - M(v)), 1 + \alpha(G - N[v])\}.$$

Proof. Vertex v can either belong to a maximum independent set or not, from which we obtain the trivial equation

$$\alpha(G) = \max\{\alpha(G - \{v\}), 1 + \alpha(G - N[v])\}.$$

Thus it is sufficient to show that, if v is not contained in any maximum independent set, the same holds for its mirrors $M(v)$. Following the proof of Lemma 2, if no maximum independent set contains v, every maximum independent set contains at least two vertices in $N(v)$. Consider a mirror $u \in M(v)$. Since every independent set contains at most one vertex in $N(v) \setminus N(u)$ (which is a clique by assumption), it must contain at least one vertex in $N(v) \cap N(u) \subseteq N(u)$. It follows that u is not contained in any maximum independent set. □

4.2 The Algorithm

Our algorithm `mis` is described in Figure 4. In the base case $|V(G)| \leq 1$, the algorithm returns the optimum solution $\texttt{mis}(G) = |V(G)|$ (line 1). Otherwise, `mis` tries to reduce the size of the problem by applying Lemma 2 and Lemma 3. Specifically, if G contains a proper connected component C (line 2), the algorithm recursively solves the subproblems induced by C and $G - C$ separately, and sums the solutions obtained

$$\texttt{mis}(G) = \texttt{mis}(C) + \texttt{mis}(G - C).$$

Else, if there are two (adjacent) vertices v and w, with $N[w] \subseteq N[v]$ (line 3), `mis` discards v:

$$\texttt{mis}(G) = \texttt{mis}(G - \{v\}).$$

Figure 4 Algorithm `mis` for the maximum independent set problem.

```
    int mis(G) {
1       if(|V(G)| ≤ 1) return |V(G)|;
2       if(∃ component C ⊂ G) return mis(C)+mis(G − C);
3       if(∃ vertices v and w: N[w] ⊆ N[v]) return mis(G − {v});
4       if(∃ a vertex v, with d(v) = 2) return 1+mis(G̃(v));
5       select a vertex v of maximum degree, which minimizes |E(N(v))|;
6       return max{mis(G − {v} − M(v)), 1+mis(G − N[v])};
    }
```

If none of the conditions above holds, and there is a (foldable) vertex v of degree two, the algorithm folds it (line 4):

$$\texttt{mis}(G) = 1 + \texttt{mis}(\widetilde{G}(v)).$$

As a last choice, the algorithm selects a vertex v of maximum degree which minimizes the number $|E(N(v))|$ of edges in its neighborhood, and branches on it according to Lemma 4 (lines 5-6):

$$\texttt{mis}(G) = max\{\texttt{mis}(G - \{v\} - M(v)), 1 + \texttt{mis}(G - N[v])\}.$$

Choosing a vertex of maximum degree for branching (line 5) is a natural "greedy" choice. The reason for choosing a vertex with few edges in its neighborhood will be clearer from the analysis. Notice that, with simple modifications, `mis` can also provide one maximum independent set (besides its cardinality).

Standard analysis. Also in this case, to underline the importance of a good choice of the measure, we sketch the analysis of `mis` according to the standard measure $k = k(G) = n$. Let $P(k)$ be the maximum number of base instances generated by the algorithm to solve a problem of size k. Of course, $P(k) = 1$ for $k \leq 1$. If the condition of line 2 is satisfied, $P(k) \leq P(k_1) + P(k - k_1)$, where k_1 is the number of vertices of C. If one of the conditions of lines 3 and 4 is satisfied, $P(k) \leq P(k-1)$ since we decrease the number of vertices in the graph at least by one. Otherwise, consider the vertex v on which we branch. Note that all the vertices in the graph must have degree at least three. Moreover v is a vertex of maximum degree. If $d(v) = 3$ (and hence the graph is 3-regular), when we discard v, we either discard a mirror of v or we fold a neighbor w of v in the following step (since $d(w) = 2$ after removing v). In both cases, we decrease the number of vertices by at least two. When we select v, we discard $N[v]$, where $|N[v]| = 4$. This leads to $P(k) \leq P(k-2) + P(k-4)$. Assume now that $d(v) \geq 4$. In the worst case, v has no mirrors ($M(v) = \emptyset$). When we discard or select v, we remove at least one or five vertices, respectively. Thus $P(k) \leq P(k-1) + P(k-5)$. We can conclude that $P(k) = O^*(\lambda^k)$, where $\lambda = 1.3247\ldots < 2^{0.406}$ is the largest root of the polynomials $x^5 - x^4 - 1$ and $x^4 - x^2 - 1$. Since in each step the size of the graphs generated decreases by at least one, it follows that the depth of the search tree is at most n. Moreover, solving each subproblem, not considering the possible recursive calls, takes polynomial time. Thus the time complexity of the algorithm is $O^*(\lambda^n) = O^*(2^{0.406\,n})$.

In the next section we will show how to refine the running time analysis of `mis` by means of a more sophisticated measure $k(G)$ (without modifying the algorithm!).

4.3 The Analysis

When we measure the size of a maximum independent set instance with the number of vertices, we do not take into account the fact that decreasing the degree of a vertex v has a positive impact on the progress of the algorithm (even if we do not immediately remove v from the graph). In fact, decreasing the degree of a vertex pays off in the long term, since the vertices of degree at most two can be filtered out without branching.

This suggests the idea of giving different weights to vertices of different degree. In particular, let n_i ($n_{\geq i}$) denote the number of vertices of degree i (at least i) in the graph considered. We will use the following measure $k = k(G)$ of the size of G:

$$k(G) = \sum_{i \geq 0} \alpha_i \, n_i,$$

where the weights $\alpha_i \in [0, 1]$ will be fixed in the following. Note that $k = k(G) \leq n$. In order to simplify the running time analysis, we make the following assumptions:

- First of all, $\alpha_0 = \alpha_1 = \alpha_2 = 0$. The reason for this assumption is that vertices of degree at most two are removed from the graph without branching in lines 1,3, and 4. Thus their presence contributes to the running time only with a polynomial (multiplicative) factor.

- Second, $\alpha_i = 1$ for $i \geq 7$. This way, we have to compute only a finite (small) number of weights.

- When the degree of a vertex decreases from i to $i - 1$, its weight decreases by $\Delta \, \alpha_i = \alpha_i - \alpha_{i-1}$. We assume that $\Delta \, \alpha_3 \geq \Delta \, \alpha_4 \geq \Delta \, \alpha_5 \geq \Delta \, \alpha_6 \geq \Delta \, \alpha_7 \geq 0$. In other words, the weights decrease from α_7 to α_2 at increasing speed. The reason for this assumption will be clearer from the analysis.

- Eventually, we impose

  $$\alpha_2 + \alpha_{d_1} + \alpha_{d_2} - \alpha_{d_1+d_2-2} = \alpha_{d_1} + \alpha_{d_2} - \alpha_{d_1+d_2-2} \geq 0, \quad \forall \, d_1, d_2 \in \{2, 3, \ldots, 8\}.$$

  This condition ensures that, when we fold a vertex of degree two, the size of the problem does not increase.

We are now ready to give our refined analysis of `mis`.

Theorem 4 *Algorithm* `mis` *solves the maximum independent set problem in* $O^*(2^{0.295\,n})$ *time and polynomial space.*

Proof. The correctness of the algorithm immediately follows from Lemmas 2, 3, and 4. The algorithm is trivially polynomial-space.

Let $P(k)$ denote the maximum number of base instances generated by `mis` when solving a problem of size k. From the discussion above, the running time of the algorithm is $O^*(P(k))$. When $k = 0$, the maximum degree in the graph is two. In such case, it is easy to see that the algorithm solves the problem in polynomial time. Hence $P(0) = O(n^{O(1)}) = O^*(1)$. Thus, let us assume $k > 0$. We break the running time analysis in different parts, one for each branching and reduction rule of the algorithm.

(1) Connected components. Let k_1 be the size of the connected component C selected by the algorithm. The size of $G - C$ is trivially $k_2 = k - k_1$. Thus

$$P(k) \leq P(k_1) + P(k - k_1). \tag{1}$$

(2) Dominance. When we remove a vertex v by dominance, we generate a unique subproblem of size $k_1 = k - \alpha_{d(v)} - \sum_{u \in N(v)} \Delta\, \alpha_{d(u)}$. Hence

$$P(k) \leq P(k_1).$$

Note that $k_1 \leq k$ by the assumptions on the weights.

(3) Folding. Let v be the vertex that we fold, and $N(v) = \{u_1, u_2\}$. Note that, by dominance, u_1 and u_2 cannot be adjacent. Thus, by folding v, we remove $N[v]$ and we introduce a unique vertex u_{12} of degree $d(u_{12}) \leq (d(u_1) - 1) + (d(u_2) - 1) = d(u_1) + d(u_2) - 2$. It follows that

$$P(k) \leq P(k_1),$$

where $k_1 = k - \alpha_2 - \alpha_{d(u_1)} - \alpha_{d(u_2)} + \alpha_{d(u_1)+d(u_2)-2}$. Also in this case, by the assumptions on the weights, $k_1 \leq k$.

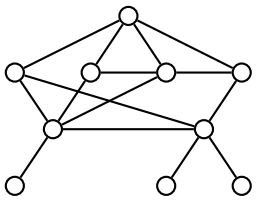
(4) Branching. Let v the vertex at which the algorithm branches. We let $d := d(v) \geq 3$ and $N(v) = \{u_1, u_2, \ldots, u_d\}$. The following properties hold:

(i) For every vertex w, $3 \leq d(w) \leq d$. Moreover, if $d(w) = d$, $|E(N(w))| \geq |E(N(v))|$;

(ii) The graph is formed by a unique connected component, and no vertex is dominated nor dominates any other vertex.

We need some extra notation (see also Figure 5).

Figure 5 Vertex v is on the top. In the example, $d = 4$, $m_4 = 1$, $m_3 = 3$, $p_2 = p_3 = 1$, $out = 5$, $in = 2$, $\overline{in} = 4$, and $sum = 13$.



- The number of vertices of degree i in $N(v)$ is m_i.
- The number of vertices in $N^2(v)$ that have exactly h neighbors in $N(v)$ is p_h.
- The number of edges between $N(v)$ and $N^2(v)$ is out.
- The number of edges and anti-edges in $N(v)$ is in and $\overline{in}$, respectively.
- The sum of the degrees of the vertices in $N(v)$ is sum.

We also let $m_{\geq i} = \sum_{j \geq i} m_j$ and $p_{\geq h} = \sum_{j \geq h} p_j$.

For the remainder of the analysis of the branching at vertex v, we distinguish two different subcases, depending on whether v has at least one mirror or not.

(A) At least one mirror. Trivially, $\sum_{i=3}^{d} m_i = d$ and $sum = d + 2\,in + out$. Observe that, by dominance, each vertex in $N(v)$ has at least one neighbor in $N^2(v)$. Therefore, $out \geq d$. In fact, by a simple parity argument, $out \geq d + (sum \ (mod\ 2))$. Moreover, by dominance, no vertex in $N(v)$ can have more than $(d-2)$ neighbors in $N(v)$. It follows that $in \leq \lfloor d(d-2)/2 \rfloor$. Summarizing the discussion above, we obtain the following constraint:

$$\sum_{i=3}^{d} m_i = d; \quad sum = d + 2\,in + out; \quad out \geq d + (sum \ (mod\ 2)); \quad in \leq \lfloor d(d-2)/2 \rfloor. \quad (2)$$

Suppose we discard v. The size of the problem decreases by α_d because of the removal of v, and by at least $\sum_{i=3}^{d} m_i\,\Delta\,\alpha_i$ because of the reduction of the degrees of vertices in $N(v)$. Note that the vertices $w \in N^2(v)$ with $|N(w) \cap N(v)| \geq d-1$ are mirrors of v. The removal of those mirrors implies a decrease of the size of the problem by $p_d\,\alpha_d + p_{d-1}\,\alpha_{\max\{3,\ d-1\}}$ (where we use the fact that the minimum degree of any vertex is at least 3). Even when $p_{d-1} + p_d = 0$, the assumption that there is at least one mirror implies that the size of the problem decreases by at least α_3. Altogether, we obtain a further decrease of the size of the problem by

$$\max\{p_d\,\alpha_d + p_{d-1}\,\alpha_{\max\{3,\ d-1\}},\ \alpha_3\}.$$

Consider now the case we select v. The size of the problem decreases by α_d because of the removal of v, and by $\sum_{i=3}^{d} m_i\,\alpha_i$ because of the removal of $N(v)$. The size of the problem further decreases because of the reduction of the degrees in $N^2(v)$. Consider a vertex $z \in N^2(v)$ with h neighbors in $N(v)$. Note that the reduction of the size of z is $\alpha_{d(z)} - \alpha_{d(z)-h}$. If $d(z) - h \geq 3$, the minimum reduction of the size of z is achieved when z has the largest possible degree $d(z) = d$:

$$\begin{aligned} \alpha_{d(z)} - \alpha_{d(z)-h} &= \Delta\,\alpha_{d(z)} + \Delta\,\alpha_{d(z)-1} + \ldots + \Delta\,\alpha_{d(z)-h+1} \\ &\geq \Delta\,\alpha_d + \Delta\,\alpha_{d-1} + \ldots + \Delta\,\alpha_{d-h+1} \\ &= \alpha_d - \alpha_{d-h}. \end{aligned}$$

Otherwise, the reduction is

$$\alpha_{d(z)} - \alpha_{d(z)-h} = \alpha_{d(z)} \geq \alpha_{max\{3,h\}},$$

where we used the fact that $d(z) \geq h$ trivially, and $d(z) \geq 3$ from the properties of the algorithm. Thus, we can pessimistically assume that the total reduction of the size of the vertices in $N^2(v)$ is at least

$$\sum_{h=1}^{d} p_h\ \min\{\alpha_{\max\{3,h\}},\ \alpha_d - \alpha_{d-h}\}.$$

Altogether, we obtain the following set of recurrences for all the m_i's and p_h's satisfying (2):

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

where

$$\Delta\,k_{OUT} = \alpha_d + \sum_{i=3}^{d} m_i\,\Delta\,\alpha_i + \max\{p_d\,\alpha_d + p_{d-1}\,\alpha_{\max\{3,\ d-1\}},\ \alpha_3\},$$

$$\Delta\,k_{IN} = \alpha_d + \sum_{i=3}^{d} m_i\,\alpha_i + \sum_{h=1}^{d} p_h\ \min\{\alpha_{\max\{3,h\}},\ \alpha_d - \alpha_{d-h}\}. \quad (3)$$

(B) No mirror. If v has no mirror, we can use the same analysis as in case (A), but removing the term $\max\{p_d\,\alpha_d + p_{d-1}\,\alpha_{\max\{3,\,d-1\}},\ \alpha_3\}$ in (3). However, in this case we can put some extra constraints on the p_h's which are crucial for reducing the final running time bound.

First of all, we observe that $p_{d-1} = p_d = 0$ since there are no mirrors by assumption. Moreover, $in \leq \lfloor d(d-2)/2 \rfloor - 1$. In fact, assume by contradiction that $in = \lfloor d(d-2)/2 \rfloor$. This implies that $\overline{in} = \lceil d/2 \rceil$. For $d = 3$, $N(v)$ contains one edge, say u_1u_2, and hence the two neighbors of u_3 in $N^2(v)$ are mirrors of v, which is a contradiction. Consider now the case $d \geq 4$. Since there are only $\lceil d/2 \rceil$ anti-edges in $N(v)$, and by dominance each vertex of $N(v)$ must be incident to at least one such anti-edge, there must be two non-adjacent vertices of $N(v)$, say u_1 and u_2, which are adjacent to all the other vertices of $N(v)$. In particular, $N(v) \setminus \{u_1, u_2\}$ contains at least $\overline{in} - 1$ anti-edges. Note that $d(u_1) = d$ by dominance, from which it follows that $N(u_1)$ contains at most $\overline{in}$ anti-edges by the choice of v. Let w be the unique neighbor of u_1 in $N^2(v)$. Since $N(u_1)$ contains the $\overline{in} - 1$ anti-edges in $\{u_3, \ldots, u_d\}$, plus the anti-edge vw, w must be adjacent to all the vertices in $\{u_1, u_3, \ldots, u_d\}$. Hence w is a mirror of v, which is a contradiction.

In fact, we can go one step further. There must be a vertex in $N(v)$, say u_1, incident to at most $\lfloor 2\,\overline{in}/d \rfloor$ anti-edges of $N(v)$. Suppose $\lfloor 2\,\overline{in}/d \rfloor \leq 1$. Then, by dominance, u_1 is incident to exactly one anti-edge of $N(v)$, say u_1u_2. Hence u_1 must be adjacent to all the vertices in $\{u_3, \ldots, u_d\}$. Moreover, it must be $d(u_1) = d$, which implies that $N(u_1)$ can contain at most $\overline{in}$ anti-edges. Let w be the neighbor of u_1 in $N^2(v)$. Note that $N(u_1)$ already contains the anti-edge wv, and at least one anti-edge in $\{u_3, \ldots, u_d\}$ (otherwise $\{u_1, u_3, \ldots, u_d\}$ would be a clique, and hence the neighbors of u_2 in $N^2(v)$ would be mirrors). There can be at most $\overline{in} - 2$ other anti-edges in $N(u_1)$. In particular, there can be at most $\overline{in} - 2$ anti-edges between w and the vertices in $\{u_3, \ldots, u_d\}$. It follows that there are at least $(d-2) - (\overline{in} - 2) + 1 = d - \overline{in} + 1$ edges between w and $\{u_1, u_3, \ldots, u_d\}$. Then $p_{\geq d - \overline{in} + 1} \geq 1$ for $\lfloor 2\,\overline{in}/d \rfloor \leq 1$. Summarizing the discussion above, we obtain the following constraint:

$$\begin{aligned} &\sum_{i=3}^{d} m_i = d; \quad sum = d + 2\,in + out; \quad out \geq d + (sum \ (mod\ 2)); \\ &in \leq \lfloor d(d-2)/2 \rfloor - 1; \quad p_{d-1} = p_d = 0; \quad \lfloor 2\,\overline{in}/d \rfloor \leq 1 \ \Rightarrow\ p_{\geq d - \overline{in} + 1} \geq 1. \end{aligned} \tag{4}$$

Altogether we get the following set of recurrences for all the m_i's and p_h's satisfying (4):

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

where

$$\begin{aligned} \Delta\,k_{OUT} &= \alpha_d + \sum_{i=3}^{d} m_i\,\Delta\,\alpha_i, \\ \Delta\,k_{IN} &= \alpha_d + \sum_{i=3}^{d} m_i\,\alpha_i + \sum_{h=1}^{d-2} p_h\ \min\{\alpha_{\max\{3,h\}},\ \alpha_d - \alpha_{d-h}\}. \end{aligned} \tag{5}$$

Limiting the number of recurrences. Consider any branching node v with $d = d(v) \geq 8$. Let d_i be the degree of the i-th neighbor of v. Observe that, in all the recurrences considered,

$$\Delta\,k_{OUT} \geq \alpha_d + \sum_{i=3}^{d} m_i\,\Delta\,\alpha_i = 1 + \sum_{i=1}^{d} \Delta\,\alpha_{\min\{d_i,8\}} \geq 1 + \sum_{i=1}^{8} \Delta\,\alpha_{\min\{d_i,8\}},$$

Table 2 The worst-case recurrences in the analysis of `mis`.

$d = 7$, $m_7 = 7$, $out = 14$, $in = 14$, $p_1 = 14$, no mirror
$d = 6$, $m_6 = 6$, $out = 10$, $in = 10$, $p_1 = 8$, $p_2 = 1$, no mirror
$d = 5$, $m_5 = 5$, $out = 8$, $in = 6$, $p_1 = 6$, $p_2 = 1$, no mirror
$d = 4$, $m_4 = 4$, $out = 6$, $in = 3$, $p_1 = 4$, $p_2 = 1$, no mirror
$d = 3$, $m_3 = 3$, $out = 4$, $in = 1$, $p_1 = 1$, $p_3 = 1$, one mirror

where we used the fact that, by assumption, $\Delta\,\alpha_i = 0$ for $i \geq 8$. Similarly,

$$\Delta\,k_{IN} \geq \alpha_d + \sum_{i=3}^{d} m_i\,\alpha_i = 1 + \sum_{i=1}^{d} \alpha_{\min\{d_i,8\}} \geq 1 + \sum_{i=1}^{8} \alpha_{\min\{d_i,8\}}.$$

As a consequence, in the case $d = d(v) \geq 8$, we can replace the Recurrences (3) and (5) with the following dominating set of recurrences: for all $d_1, d_2, \ldots, d_8 \in \{3, 4, \ldots, 8\}$,

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

where

$$\Delta\,k_{OUT} = 1 + \sum_{i=1}^{8} \Delta\,\alpha_{d_i} \quad \text{and} \quad \Delta\,k_{IN} = 1 + \sum_{i=1}^{8} \alpha_{d_i}. \tag{6}$$

This way, we have to consider a finite (though very large) number of recurrences only. (The actual number of recurrences is 4793253).

Computing the weights. By solving the set of recurrences above, one obtains $P(k) = O^*(\lambda^k)$, where $\lambda \geq 1$ is a (quasi-convex) function of the weights. Using the same kind of approach as in Section 3.2, we numerically found that, for

$$(\alpha_3, \alpha_4, \alpha_5, \alpha_6) = (0.620196, 0.853632, 0.954402, 0.993280),$$

$\lambda \leq 2^{0.295}$. This gives a running time of $O^*(P(k)) = O^*(2^{0.295\,n})$.

The tight (branching) recurrences for this choice of the weights are indicated in Table 2. In the Appendix (Figure 10) we provide the pseudo-code of a program which can be used to check the claimed value of λ for the set of weights considered.

□

4.4 Refining the Analysis

The running time established in the previous subsection can be refined via a more careful case analysis. In principle, one might enumerate all the feasible local configurations up to some small distance h from the branching vertex v, and compute the decrease of the size of the problem when v is selected and discarded, respectively. The problem is that the number of feasible configurations is huge already for $d(v) = 7$ and $h = 2$. Giving such kind of refined analysis is out of the scope of this paper. However, in order to show the kind of improvements which are achievable (without modifying the algorithm and measure), we prove the following refined time bound on `mis`.

Theorem 5 *Algorithm* `mis` *solves the maximum independent set problem in* $O^*(2^{0.287\,n})$ *time and polynomial space.*

Since the proof of the theorem above is technical, and does not introduce any substantially new idea, we give it in the Appendix.

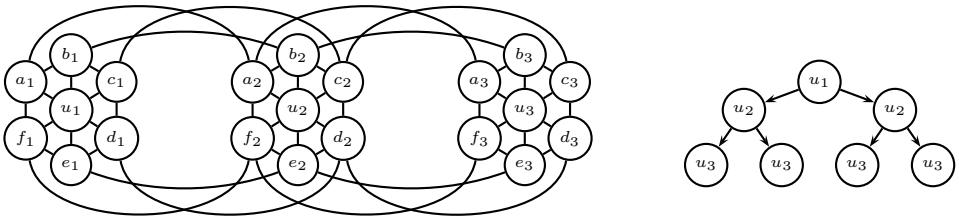
4.5 An Exponential Lower Bound

As in the case of `mds`, it makes sense to search for a lower bound on the worst-case running time of `mis` to see how far we are from a tight analysis.

Theorem 6 *The worst-case running time of algorithm* `mis` *is* $\Omega(2^{n/7}) = \Omega(2^{0.142\,n})$.

Proof. Consider the following connected graph G_ℓ, $\ell \geq 1$, of $n = 7\ell$ vertices: G_ℓ consists of ℓ blocks $B_1, B_2, \ldots, B_\ell$. Each block B_i, $1 \leq i \leq \ell$, is a 6-wheel, i.e. it is formed by six nodes a_i, b_i, c_i, d_i, e_i, and f_i which induce a chord-less cycle, and by a vertex u_i which is adjacent to all the vertices in the cycle. For each $i = 1, \ldots, \ell - 1$, graph G_ℓ also contains edges $\{a_i, a_{i+1}\}$, $\{b_i, b_{i+1}\}$, $\{c_i, c_{i+1}\}$, $\{d_i, d_{i+1}\}$, $\{e_i, e_{i+1}\}$, and $\{f_i, f_{i+1}\}$. (See Figure 6 for an example).

Figure 6 On the left, graph G_ℓ for $\ell = 3$. On the right, the top part of a feasible search tree: there is a vertex in the tree for each subproblem; subproblems are labelled with the corresponding branching vertex; left and right children correspond to selection and discarding of the branching vertex, respectively.



Let us apply Algorithm `mis` to graph G_ℓ. Note that, there is a unique connected component and no dominance nor folding can be applied. Hence the algorithm branches at some vertex of maximum degree, with minimum number of edges in its neighborhood. In particular, `mis` might branch at u_1. In the subproblems where u_1 is selected, we remove the vertices of B_1 arriving at a graph $G_{\ell-1}$. By the same argument as above, in that subproblem `mis` might branch at u_2. Also in the subproblem where u_1 is discarded, the algorithm cannot apply dominance nor folding. Thus also in that case u_2 is a feasible candidate for branching. Therefore, in both subproblems the algorithm might branch at u_2. By iterating this argument, one finds that the algorithm might branch on the ordered sequence $u_1, u_2, \ldots, u_\ell$. We conclude that the running time of the algorithm is $\Omega(2^{n/7}) = \Omega(2^{0.142\,n})$. □

5 Conclusions and Future Work

In this paper we investigated the impact of non-standard measures in the analysis of Branch & Reduce algorithms. Using the minimum dominating set and the maximum independent set problems as case studies, we showed that the choice of the measure can have a tremendous

impact on the running time bounds achievable. This suggests the possibility that finding a good measure should be at first concern when designing Branch & Reduce algorithms.

The measures considered in this paper are still reasonably simple, though they already involve a quite large number of variables. However, there is no limit to the kind of measures that can be exploited. For example, the authors of this paper applied Measure & Conquer to design the first algorithm for the *connected dominating set* problem faster than trivial enumeration [22]. That result is based on a new measure, that considers, besides cardinalities and frequencies, also the local connectivity properties of the original graph. The already mentioned measure by Eppstein for cubic-TSP [16] is another good example of how non-trivial measures can help in the analysis. In fact, we believe that the design of new measures can have an impact comparable to (and sometimes larger than) the design of better branching and reduction rules.

Since the appearance of the preliminary versions of this article [19, 20], the Measure & Conquer technique has been turned into a common tool used in the analysis of exact graph algorithms and it was used for a variety of problems. For example, Gupta et al. [32] used the technique while analyzing exact algorithms for finding maximal induced subgraphs of fixed vertex degrees. Razgon [54], using a non-standard measure, derived the first non-trivial algorithm breaking the $O^*(2^n)$ barrier for the *feedback vertex set* problem (see also [18]). Kowalik [46] used Measure & Conquer in his branching algorithm for the edge coloring problem. The analysis of Gasper-Liedloff's algorithm for the *independent dominating set* problem in [28] is based on Measure & Conquer. Another example is the paper by Kratsch and Liedloff on the *minimum dominating clique* problem [47]. We are also aware of a number of other (still unpublished) papers using the same kind of approach.

Measure & Conquer can be used also as a tool to prove tighter combinatorial bounds. For example, using this kind of approach and the same measure which is used here for `MDS`, Fomin et al. [23] proved that the number of minimal dominating sets in a graph is $O^*(2^{0.783\,n})$. Based on this result, they also derived the first non-trivial exact algorithms for the *domatic number* problem and for the *minimum-weight dominating set* problem (see also [4, 26, 44, 58]). The bounds on the number of minimal feedback vertex sets (or maximal induced forests) obtained in [18] are also based on Measure & Conquer.

Of course, a non-standard measure can be used to design better algorithms in the standard way: one considers the tight recurrences for a given algorithm (and measure), and tries to design better branching and reduction rules for the corresponding cases. A very recent work by van Rooij and Bodlaender goes in this direction [65].

For the reasons above, we think Measure & Conquer might have a growing impact in the field of exact algorithms in the next few years.

Acknowledgements. We are grateful to Jianer Chen, Johan van Rooij, Saket Saurabh, Magnus Wahlström, and Philipp Zumstein for discussions, comments and remarks on the preliminary versions of this paper.

References

[1] M. Alekhnovich, E. Hirsch, and D. Itsykon. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. *Journal of Automated Reasoning*, 35(1-3):51-72, 2005.

[2] R. Beigel. Finding maximum independent sets in sparse and general graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 856–857, 1999.

[3] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005.

[4] A. Björklund and T. Husfeldt. Inclusion-exclusion algorithms for counting set partitions. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 575–582, 2006.

[5] H. Bodlaender, F. V. Fomin, A. Koster, D. Kratsch, and D. Thilikos. Exact algorithms for treewidth. In *European Symposium on Algorithms (ESA)*, pages 672–683, 2006.

[6] T. Brueggemann and W. Kern. An improved deterministic local search algorithm for 3-SAT. *Theoretical Computer Science*, 329:303–313, 2004.

[7] J. M. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, 2004.

[8] J. Chen, I. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.

[9] J. Chen, I. A. Kanj, and G. Xia. Labeled search trees and amortized analysis: improved upper bounds for NP-hard problems. *Algorithmica*, 43(4):245–273, 2005.

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

[11] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, 5:394–397, 1962.

[12] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[13] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer-Verlag, 1999.

[14] C. Ebenegger, P. L. Hammer, and D. de Werra. Pseudo-boolean functions and stability of graphs. *Annals of Discrete Mathematics*, 19:83–98, 1984.

[15] D. Eppstein. Small maximal independent sets and faster exact graph coloring. *Journal of Graph Algorithms and Applications*, 7(2):131–140, 2003.

[16] D. Eppstein. The Traveling Salesman Problem for Cubic Graphs. *Journal of Graph Algorithms and Applications*, 11(1):61–81, 2007.

[17] D. Eppstein. Quasiconvex analysis of backtracking algorithms. *ACM Transactions on Algorithms*, 2(4):492–509, 2006.

[18] F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon. On the Minimum Feedback Vertex Set Problem: Exact and Enumeration Algorithms. *Algorithmica*, 52(2):293–307, 2008.

[19] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: domination - a case study. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 191–203, 2005.

[20] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: a simple $O(2^{0.288\,n})$ independent set algorithm. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 18–25, 2006.

[21] F. V. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the European Association for Theoretical Computer Science*, 87:47–77, 2005.

[22] F. V. Fomin, F. Grandoni, and D. Kratsch. Solving connected dominating set faster than 2^n. *Algorithmica,* 52(2):153–166, 2008.

[23] F. V. Fomin, F. Grandoni, A. Pyatkin, and A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms,* 5(1): 2008.

[24] F. V. Fomin, D. Kratsch, I. Todinca, and Y. Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM Journal Computing*, 38(3):1058-1079, 2008.

[25] F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 199–210, 2004.

[26] F. V. Fomin and A. Stepanov. Counting minimum weighted dominating sets. In *International Computing and Combinatorics Conference (COCOON)*, pages 65–74. 2007.

[27] M. Fürer. A faster algorithm for finding maximum independent sets in sparse graphs. In *Latin American Theoretical Informatics Symposium (LATIN)*, pages 491–501, 2006.

[28] S. Gaspers and M. Liedloff. A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. In *Graph-Theoretic Concepts in Computer Science (WG)*, pages 78–89, 2006.

[29] R. N. J. Gramm, E. A. Hirsch and P. Rossmanith. Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT. *Discrete Applied Mathematics*, 130(2):139–155, 2003.

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

[31] F. Grandoni. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms*, 4(2):209–214, 2006.

[32] S. Gupta, V. Raman, and S. Saurabh. Fast exponential algorithms for maximum -regular induced subgraph problems. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 139–151. 2006.

[33] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999.

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

[35] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, 10:196–210, 1962.

[36] E. A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.

[37] E. A. Hirsch. SAT local search algorithms: worst-case study. *Journal of Automated Reasoning*, 24(1-2):127–143, 2000.

[38] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the Association for Computing Machinery*, 21:277–292, 1974.

[39] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponetial complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

[40] K. Iwama. Worst-case upper bounds for k-SAT. *Bulletin of the European Association for Theoretical Computer Science*, 82:61–71, 2004.

[41] K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 328–329, 2004.

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

[43] S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the traveling salesman problem. In *Proceedings of the annual ACM conference*, pages 294–300, 1977.

[44] M. Koivisto. An $O(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 583–590, 2006.

[45] A. Kojevnikov and A. S. Kulikov. A new approach to proving upper bounds for MAX-2-SAT. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 11–17, 2006.

[46] L. Kowalik. Improved edge-coloring with three colors. In *Graph-Theoretic Concepts in Computer Science (WG)*, pages 90–101. 2006.

[47] D. Kratsch and M. Liedloff. An exact algorithm for the minimum dominating clique problem. *Theoretical Computer Science,* 385(1-3):226–240, 2007.

[48] O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1-2):1–72, 1999.

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

[50] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.

[51] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *Journal of the Association for Computing Machinery*, 52(3):337–364, 2005.

[52] P. Pudlak and R. Impagliazzo. A lower bound for DLL algorithms for k-SAT. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 128–136, 2000.

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

[54] I. Razgon. Exact computation of maximum induced forest. In *Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 160–171, 2006.

[55] I. Razgon. A faster solving of the maximum independent set problem for graphs with maximal degree 3. In *Algorithms and Complexity in Durham Workshop (ACiD)*, pages 131–142, 2006.

[56] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP. In *ACM Symposium on the Theory of Computing (STOC)*, pages 475–484, 1997.

[57] B. Reed. Paths, stars and the number three. *Combinatorics, Probability and Computing*, 5(3):277–295, 1996.

[58] T. Riege, J. Rothe, H. Spakowski, and M. Yamamoto. An improved exact algorithm for the domatic number problem. *Information Processing Letters*, 101(3):101–106, 2007.

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

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

[61] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 410–414, 1999.

[62] U. Schöning. Algorithmics in exponential time. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 36–43, 2005.

[63] R. Schroeppel and A. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal on Computing*, 10(3):456–464, 1981.

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

[65] J. van Rooij and H. L. Bodlaender. Design by Measure and Conquer, A Faster Exact Algorithm for Dominating Set. In *Symposium on Theoretical Aspects of Computer Science,* pages 657–668, 2008.

[66] Y. Villanger. Improved exponential-time algorithms for treewidth and minimum fill-in. In *Latin American Theoretical Informatics Symposium (LATIN)*, pages 800–811, 2006.

[67] D. B. West. Introduction to Graph Theory. *Prentice Hall*, 1996.

[68] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348:2–3, 2005.

[69] G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *International Workshop on Combinatorial Optimization – Eureka, You Shrink*, number 2570 in Lecture Notes in Computer Science, pages 185–207. Springer-Verlag, 2003.

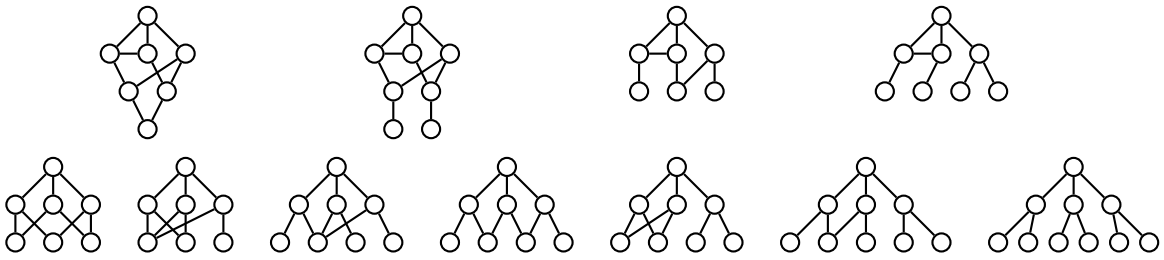
[70] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *ACM Symposium on the Theory of Computing (STOC)*, pages 681–690, 2006.

Appendix

Proof. **(Theorem 5)** We refine the analysis of Theorem 4. Consider a branching step, and let v be the corresponding branching vertex. Recall that, when we branch, the graph is formed by a unique connected component. We observe the following facts:

(a) $\mathbf{d} = \mathbf{d}(\mathbf{v}) = \mathbf{3}$. If the number n of vertices in the graph is small, say $n \leq 6$, the algorithm trivially solves the problem in polynomial time. Hence $P(k) = O^*(1)$ in that case. As a consequence, without loss of generality, we can assume that the graph contains at least seven vertices. Observe that all the vertices of the graph have degree three. Thus by dominance $in \leq 1$. Suppose $N(v)$ contains one edge (that is, $in = 1$), say u_1u_2. If this is the case, let w_1 and w_2 be the two neighbors of u_3 in $N^2(v)$ (which are mirrors). By dominance, u_1 and u_2 cannot be adjacent to the same vertex in $N^2(v)$. In particular, it must be the case that $p_3 = 0$. Moreover, w_1 and w_2 cannot be adjacent when $p_2 = 2$ (otherwise the graph would contain six vertices). Consider now the case $in = 0$. Note that, by the same argument as above, it cannot be $p_3 = 2$. The feasible remaining local configurations around v are represented in Figure 7. Note that, when we discard v (and its mirrors), or we select it, some vertices in $N(v) \cup N^2(v)$

Figure 7 Feasible local configurations for $d = 3$. Vertex v is on the top.



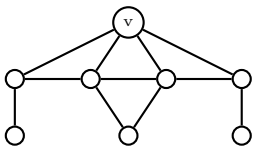
might be selected or discarded by dominance. In turn, this can determine the removal or the decrease of the degree of some other vertices. By considering the different cases one by one, it is not hard to obtain the following set of recurrences:

$$\begin{aligned} P(k) &\leq P(k - 4\,\alpha_3) + P(k - 10\,\alpha_3), \\ P(k) &\leq P(k - 6\,\alpha_3) + P(k - 8\,\alpha_3), \\ P(k) &\leq P(k - 7\,\alpha_3) + P(k - 7\,\alpha_3). \end{aligned} \tag{7}$$

Recurrences (7) can replace the less accurate Recurrences (3) and (5) for $d = 3$:

(b) $\mathbf{d = d(v) = 4}$, $\mathbf{in = 3}$, and $\mathbf{M(v) = \emptyset}$. The edges in $N(v)$ cannot induce a triangle, say $\{u_1, u_2, u_3\}$, since otherwise the neighbors of u_4 in $N^2(v)$ would be mirrors of v. Hence, $N(v)$ must induce a path, say u_1, u_2, u_3, u_4. Observe that it must be the case that $d(u_2) = d(u_3) = 4$ by dominance. Let w_1 be the neighbor of u_2 in $N^2(v)$. Note that, by the way v is chosen, $N(u_2)$ can contain at most three anti-edges. Since $N(u_2)$ contains the anti-edges vw_1 and u_1u_3, we have that w_1 must be adjacent to one between u_1 and u_3. In fact, w_1 must be adjacent to u_3 and not adjacent to u_1, since otherwise w_1 would be a mirror of v. Assume by contradiction that $d(u_1) = 4$, and hence $N(u_1)$ contains at most three anti-edges. Let w' and w'' be the two neighbors of u_1 in $N^2(v)$ (distinct from w_1). Then $N(u_1) = \{v, u_2, w', w''\}$ contains the anti-edges vw', vw'', u_3w', and u_3w'', which is a contradiction. Thus $d(u_1) = 3$. A symmetric argument shows that $d(u_4) = 3$. If u_1 and u_4 had a common neighbor $w' \in N^2(v)$, w' would be a mirror of v, contradicting the assumption. Hence we can assume that u_1 and u_4 have two distinct neighbors w_2 and w_3, respectively, in $N^2(v)$. (See Figure 8). Now we

Figure 8 The unique feasible local configuration for $d = 4$, $in = 3$, and $M(v) = \emptyset$.



distinguish two subcases, depending on the degree of w_2 and w_3. If they have both degree three, we obtain the following recurrence:

$$\begin{aligned} P(k) \quad &\leq P(k - \alpha_4 - 2\Delta\,\alpha_3 - 2\Delta\,\alpha_4) + P(k - \alpha_4 - 2\,\alpha_3 - 2\,\alpha_4 - (\alpha_4 - \alpha_2) - 2\Delta\,\alpha_3) \\ &= P(k - 3\,\alpha_4) + P(k - 4\,\alpha_4 - 4\,\alpha_3). \end{aligned} \tag{8}$$

Observe that the decrease of the degree of w_2 and w_3 when v is selected contributes with a term $-2\Delta\,\alpha_3$. According to Recurrence (5), the corresponding term is $-2\min\{\alpha_3, \Delta\,\alpha_4\} = -2\Delta\,\alpha_4$ only. Suppose now that one among w_2 and w_3, say w_2, has degree four. It follows by the way v is chosen that $N(w_2)$ contains at most three anti-edges. Let z_1, z_2, and z_3 be the remaining neighbors of w_2, besides u_1. Note that the z_i's must belong to $N^2(v) \cup N^3(v)$. By the local structure of the graph, no z_i can be adjacent to u_1, that is z_1u_1, z_2u_1, and z_3u_1 are anti-edges. It follows that the z_i's induce a clique. By dominance with w_2, this also implies that they have degree four. In the case v is selected, and hence u_1 is discarded, by dominance with w_2 one of the z_i is discarded. This gives the following recurrence

$$\begin{aligned} P(k) \quad &\leq P(k - \alpha_4 - 2\Delta\,\alpha_3 - 2\Delta\,\alpha_4) + P(k - \alpha_4 - 2\,\alpha_3 - 2\,\alpha_4 - (\alpha_4 - \alpha_2) - 2\Delta\alpha_4 - \alpha_4) \\ &= P(k - 3\,\alpha_4) + P(k - 7\,\alpha_4). \end{aligned} \tag{9}$$

Recurrences (8) and (9) can replace the less accurate Recurrence (5) in the case considered.

Recomputing the weights. Recomputing the weights according to the refined recurrences above, we get

$$(\alpha_3, \alpha_4, \alpha_5, \alpha_6) = (0.545340, 0.811103, 0.933335, 0.985228),$$

Table 3 The worst-case recurrences in the refined analysis of `mis`.

$d = 7$, $m_7 = 7$, $out = 14$, $in = 14$, $p_1 = 14$, no mirror
$d = 6$, $m_6 = 6$, $out = 10$, $in = 10$, $p_1 = 8$, $p_2 = 1$, no mirror
$d = 5$, $m_5 = 5$, $out = 8$, $in = 6$, $p_1 = 6$, $p_2 = 1$, no mirror
$d = 4$, $m_3 = 4$, $out = 4$, $in = 2$, $p_1 = 4$, no mirror
$d = 4$, $m_4 = 4$, $out = 4$, $in = 4$, $p_1 = 1$, $p_3 = 1$, one mirror

and hence a running time of $O^*(2^{0.287\,n})$. The new tight recurrences are given in Table 3.
□

Figure 9 C-like pseudo-code to check a value of λ for given (feasible) values of the weights α_i's and β_j's for Algorithm `msc`: the function returns `true` if $1 \geq \lambda^{-\Delta\, k_{OUT}(t)} + \lambda^{-\Delta\, k_{IN}(t)}$ for every feasible tuple t, and `false` otherwise.

```
boolean checkMDS(α2, α3, α4, α5, β2, β3, β4, β5, λ){
    α1=β1=0; α6=α7=β6=β7=1;
    compute Δα_i and Δβ_i for 2 ≤ i ≤ 7;
    for(|S|=3,4,..., 7)
      for(r2=0,1,..., |S|)
        for(r3=0,1,..., |S| − r2)
          for(r4=0,1,..., |S| − r2 − r3)
            for(r5=0,1,..., |S| − r2 − r3 − r4)
              for(r6=0,1,..., |S| − r2 − r3 − r4 − r5)
                r≥7 = |S| − r2 − r3 − r4 − r5 − r6;
                compute Δ k'_{|S|,r2};
                Δ k_OUT = α_{|S|} + Σ_{i=2}^{6} r_i Δ β_i + Δ k'_{|S|,r2};
                Δ k_IN = α_{|S|} + Σ_{i=2}^{6} r_i β_i + r≥7 + Δ α_{|S|}(Σ_{i=2}^{6}(i − 1)r_i + 6 r≥7);
                if(λ^{−Δ k_OUT} + λ^{−Δ k_IN} > 1) return false;
    return true; //All the inequalities satisfied
}
```

Figure 10 C-like pseudo-code to check a value of λ for given (feasible) values of the weights α_i's for Algorithm `mis`: the function returns `true` if $1 \geq \lambda^{-\Delta k_{OUT}} + \lambda^{-\Delta k_{IN}}$, where Δk_{OUT} and Δk_{IN} are given by Recurrences (3), (5), and (6), and `false` otherwise. We recall that **continue** is used to skip the remaining part of the block considered.

```
boolean checkMIS(α3, α4, α5, α6, λ){
  α2=0; α7 = α8 = 1;
  compute Δαi for 3 ≤ i ≤ 8;
  for(d=3,4,..., 7) //Case d≤ 7
    for(i = 3, 4, ..., 7) //zi's used to set some variables to 0
      if(i ≤ d) zi = 1; else zi = 0;
    for(m7=0,1,..., z7 d)
      for(m6=0,1,..., z6 (d − m7))
        for(m5=0,1,..., z5 (d − m7 − m6))
          for(m4=0,1,..., z4 (d − m7 − m6 − m5))
            m3 = d − m7 − m6 − m5 − m4;
            for(in=0,1,..., ⌊d(d − 2)/2⌋)
              sum = Σ_{i=3}^{7} mi; in‾ = d(d − 1)/2 − in; out = sum − d − 2 in;
              if(out < d + (sum (mod 2))) continue;
              for(p7=0,1,..., z7⌊out/7⌋)
                for(p6=0,1,..., z6⌊(out − 7 p7)/6⌋)
                  for(p5=0,1,..., z5⌊(out − 7 p7 − 6 p6)/5⌋)
                    for(p4=0,1,..., z4⌊(out − 7 p7 − 6 p6 − 5 p5)/4⌋)
                      for(p3=0,1,..., z3⌊(out − 7 p7 − 6 p6 − 5 p5 − 4 p4)/3⌋)
                        for(p2=0,1,..., ⌊(out − 7 p7 − 6 p6 − 5 p5 − 4 p4 − 3 p3)/2⌋)
                          p1=out − 7 p7 − 6 p6 − 5 p5 − 4 p4 − 3 p3 − 2 p2;
                          //Case of at least one mirror
                          Δ k_OUT = α_d + Σ_{i=3}^{d} mi Δ αi + max{p_d α_d + p_{d−1} α_{max{3,d−1}}, α3};
                          Δ k_IN = α_d + Σ_{i=3}^{d} mi αi + Σ_{h=1}^{d} ph min{α_{max{3,h}}, α_d − α_{d−h}};
                          if(λ^{−Δ k_OUT} + λ^{−Δ k_IN} > 1) return false;
                          //Case of no mirror
                          if(in>⌊d(d − 2)/2⌋−1 or p_d ≠ 0 or p_{d−1} ≠ 0
                              or (⌊2in‾/d⌋≤1 and p_{≥d−in‾+1}=0)) continue;
                          Δ k_OUT = α_d + Σ_{i=3}^{d} mi Δ αi;
                          Δ k_IN = α_d + Σ_{i=3}^{d} mi αi + Σ_{h=1}^{d−2} ph min{α_{max{3,h}}, α_d − α_{d−h}};
                          if(λ^{−Δ k_OUT} + λ^{−Δ k_IN} > 1) return false;
  for(d1=3,4,...,8) //Case d≥ 8
    for(d2=3,4,...,8)
      for(d3=3,4,...,8)
        for(d4=3,4,...,8)
          for(d5=3,4,...,8)
            for(d6=3,4,...,8)
              for(d7=3,4,...,8)
                for(d8=3,4,...,8)
                  Δ k_OUT = 1 + Σ_{i=1}^{8} Δ α_{di};
                  Δ k_IN = 1 + Σ_{i=1}^{8} α_{di};
                  if(λ^{−Δ k_OUT} + λ^{−Δ k_IN} > 1) return false;
  return true; //All the inequalities are satisfied
}
```