Some new techniques in design and analysis of exact (exponential) algorithms

Fedor V. Fomin* Fabrizio Grandoni† Dieter Kratsch‡

September 5, 2005

Abstract

This survey concerns techniques in design and analysis of algorithms that can be used to solve NP hard problems faster than exhaustive search algorithms (but still in exponential time). We discuss several of such techniques: Measure & Conquer, Exponential Lower Bounds, Bounded Tree-width, and Memorization. We also consider some extensions of the mentioned techniques to parameterized algorithms.

1 Introduction

In this survey we use the term *exact algorithms* for algorithms that find exact solutions of NP-hard problem (and thus run in exponential time).

The design of exact algorithms has a long history dating back to Held and Karp's paper [42] on the travelling salesman problem in the early sixties. The last years have seen an emerging interest in constructing exponential time algorithms for combinatorial problems like Coloring [9, 14], Max-Cut [64], 3-SAT [12, 22] (see also the

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway, fomin@ii.uib.no. Supported by Norges forskningsråd project 160778/V30. This work was done while the first author was at the Humboldt-Unoversität Berlin, supported by Alexander von Humboldt Foundation.

†Dipartimento di Informatica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy, grandoni@di.uniroma1.it. Supported by Web-Minds project of the Italian Ministry of University and Research, under the FIRB program.

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

survey of Iwama [43] devoted to exponential algorithms for 3-SAT), Minimum Dominating Set [32], Tree-width [34]. There are two nice surveys of Woeginger [65, 66] describing the main techniques that have been established in the field. We also recommend the paper of Schöning [61] for an introduction to exponential time algorithms.

In this paper we review four techniques for the design and analysis of exact algorithms which were not covered in the mentioned surveys. We also show how some of the techniques can be extended to parameterized algorithms.

The techniques are

- **Measure & Conquer.** For more than 30 years Davis-Putnam-style exponential time search tree algorithms have been the most common tools used for finding exact solutions of NP-hard problems. Despite of that, the way to analyze such recursive algorithms is still far from producing tight worst case running time bounds. The "Measure & Conquer" approach is one of the recent attempts to step beyond such limitations. It is based on the choice of the measure of the subproblems recursively generated by the algorithm considered; this measure is used to lower bound the progress made by the algorithm at each branching step. A good choice of the measure can lead to a significantly better worst case time analysis. We exemplify the approach by showing how to use it for solving the Minimum Dominating Set problem.
- **Lower bounds.** Since it is so difficult to obtain tight worst case running time bounds on exponential-time search tree algorithms, the natural question, we believe, that should be addressed is to find lower bounds for the worst case running time of such algorithms. [1]
- **Bounded tree-width and dynamic programming.** Dynamic programming is another common tool for exact algorithms. Here we discuss how structural properties of graphs and combinatorial bounds can be used to obtain fast exact algorithms on planar and sparse graphs. We also show how this technique can be used for parameterized algorithms.
- **Memorization.** This technique was introduced by Robson and is used to reduce the running time of exponential-time algorithms at the cost of space. We overview this technique and explain how to use it for parameterized algorithms.

[1] Let us remark that we are interested in exponential lower bounds for a specific class of *algorithms*, so these type of results do not imply that P$\neq$ NP.

2 Measure & Conquer

In this section we study the analysis of search tree algorithms. Search tree algorithms are also called branch-and-reduce algorithms, splitting algorithms, backtracking algorithms etc. Such an algorithm is recursively applied to a problem instance and uses two types of rules. *Reduction rules* are used to simplify the instance. *Branching rules* are used to solve the problem by recursively calling smaller instances of the problem. An execution of such an algorithm can best be analyzed by a search tree: assign the root node to the input of the problem; recursively assign a child to a node for each smaller instance reached by a branching rule at the instance of the node. Our goal is to analyze the running time of search tree algorithms, i.e. to upper bound the number of nodes of the search tree in the worst case.

In [65] among the major techniques to construct exponential-time algorithms Woeginger lists "pruning the search tree" and describes the classical method to analyze search tree algorithms for the problems Independent Set, 3-SAT and Bandwidth. The analysis of such recursive algorithms is based on the bounded search tree technique: a measure of the size of an instance of the problem is defined; this measure is used to lower bound the progress made by the algorithm at each branching step. One obtains a linear recurrence or a collection of linear recurrences for each reduction and branching rule. Those linear recurrences can be solved using standard techniques. Finally the worst case is taken over all linear recurrences and a running time of the type $\mathcal{O}(\alpha^p)$ is obtained, where p is some (natural) parameter for the size of the problem.

For the last 30 years the research on exact algorithms has been mainly focused on the design of more and more sophisticated algorithms. However, measures used in the analysis of search tree algorithms had been usually very simple, e.g. number of vertices for graphs and number of variables for satisfiability problems. Retrospective it is somewhat surprising that almost all analysis of search tree algorithms used standard measures for such a long period. Although a few papers used non-standard measures the general potential of a careful choice of the measure had not been discovered until very recently.

The idea behind Measure & Conquer is to focus on the choice of the (non-standard) measure, instead of creating algorithms with more and more rules. If the measure fulfils the following three conditions then the approach outlined above works.

- The measure of an instance of a subproblem obtained by a reduction or a branching rule must be smaller than the measure of the instance of the original problem.

- The measure of each instance is nonnegative.
- The measure of the input is upper bounded by some function of "natural parameters" of the input.

The last property is needed to retranslate the asymptotic upper bound in terms of the non-standard measure into an upper bound in terms of some natural parameters for the size of the input (such as the number of vertices in a graph or the number of variables in a formula). This way one is able to derive from different (and often complicated) measures, results that are easy to state and compare.

2.1 Eppstein's work

It seems that Eppstein was the first who observed the power of using non-standard measures for analyzing search tree algorithms. He used this type of analysis in several papers, among them [9, 30].

Eppstein's TSP algorithm [30]. There is a well-known dynamic programming $\mathcal{O}(2^n \cdot n^2)$[2] algorithm for the TRAVELLING SALESMAN PROBLEM (TSP) by Held and Karp [42] and there is no improvement since 1962. Eppstein studied TSP for graphs of maximum degree three (for which the problem remains NP-hard) and obtained an $\mathcal{O}(2^{n/3}n^{O(1)})$ algorithm [30]. More precisely, he studies the TRAVELLING SALESMAN PROBLEM WITH FORCED EDGES. The input is a (multi)graph G, a cost function $c : E(G) \to \mathbb{R}$ and a set of forced edges $F \subseteq E(G)$; the output is a minimum cost hamiltonian cycle of G containing all edges of F.

The search tree algorithm is simple. It consists of various reduction rules (step 1 in [30]), a unique branching rule (step 3) and it terminates in a leaf (step 2) if $G - F$ forms a collection of disjoint 4-cycles since in this case a minimum cost solution can be computed in polynomial time. In step 3 an edge xy is chosen and then the algorithm branches in the instances $G, F \cup \{xy\}$ (force xy) and $G - xy, F$ (discard xy). The analysis of the algorithm uses the following interesting non-standard measure: $s(G, F) = |V(G)| - |F| - |C|$, where C is the set of 4-cycles of G that form connected components of $G - F$. Note that despite the negative coefficients in the definition of the measure $0 \leq s(G, F) \leq n$ for all instances G, F. Using this measure the analysis gets fairly easy.

Beigel and Eppstein's 3-coloring algorithm [9]. The $\mathcal{O}(2^{0.411n})$ time 3-coloring algorithm presented in [9] is the fastest one known. To a large extent the paper studies special CONSTRAINT SATISFACTION

[2]Whether not specified differently, n and m denote the number of vertices and edges of the input graph, respectively.

PROBLEMS (CSP). An instance of CSP consists of a collection of n variables, each with a list of possible colors, and a collection of m constraints consisting of a tuple of variables and a color for each variable. A solution assigns a possible color to each variable such that no constraint is satisfied, i.e. not every variable of the constraint is colored in the way specified by the constraint. For an instance of the problem (a, b)-CSP, each variable has at most a possible colors and each constraint involves at most b variables. Note that 3-SAT is equivalent to (2,3)-CSP. Furthermore 3-COLORING, 3-LIST-COLORING and 3-EDGE-COLORING can be translated to (3,2)-CSP.

An $\mathcal{O}(2^{0.449n})$ time algorithm for (3,2)-CSP is the fundamental result of [9]. The algorithm also solves (4,2)-CSP and then its running time is $\mathcal{O}(2^{0.854n})$. The basic idea is that any (4,2)-CSP instance can be transformed into a (3,2)-CSP instance by expanding each of its four-color variables to two three-color variables. Therefore the natural measure of a (4,2)-CSP instance would be $n = n_3 + 2n_4$, where n_i denotes the number of variables with i possible colors. Crucial for the analysis of the algorithm is the use of the non-standard measure $n = n_3 + (2 - \epsilon)n_4$ where the best choice of ϵ turns out to be $\epsilon \approx 0.095543$.

Eppstein's quasiconvex analysis [31]. Multivariate recurrences frequently arise in the analysis of the worst-case running time of search tree algorithms. Two examples are provided in the paper. One is a subroutine, used in a graph coloring algorithm [29], listing all maximal independent sets of size at most k. In fact when analyzing a search tree algorithm an instance is often characterized by more than one size parameter (variable), and thus it is convenient to establish multivariate recurrences (instead of linear recurrences based on a unique variable) for the reduction and branching rules. Those variables are part of the input or come up during an execution of the algorithm in a natural way or might be chosen to improve the upper bound of the worst-case running time to be obtained. For example, the linear recurrences in terms of $s(G, F)$ obtained in the analysis of the TSP algorithm in [30] can easily be translated into multivariate recurrences in the variables $|V(G)|$, $|F|$ and $|C|$. Furthermore the linear recurrences in terms of the non-standard measure $n = n_4 + (2 - \epsilon)n_3$ obtained for the reduction and branching rules of the (4,2)-CSP algorithm in [9] can be translated easily into multivariate recurrences in the variables n_3 and n_4.

Given the multivariate recurrences we would like to obtain an upper bound on the running time of the algorithm. Eppstein showed that this multivariate system can be turned into an *equivalent* system of recurrences in a unique variable, where the new variable is a linear combination of the size parameters. It is sufficient to choose the co-

efficients (weights) which minimize the resulting running time. The optimal weight vector can be computed using quasiconvex programming.

Byskov and Eppstein's maximal bipartite subgraph listing algorithm [16]. The $\mathcal{O}(2^{0.826n})$ time algorithm to list all maximal bipartite subgraphs of a graph is the fastest one known. The algorithm can also be found in Byskov's Ph.D. thesis [15] which contains a variety of exponential-time algorithms.

The key operations of the algorithm are: coloring a vertex black (resp. white), remove an edge and remove a vertex. The key idea is that all neighbors of a black (resp. white) vertex can either be white (resp. black) or have to be removed. To indicate this state they will be half-colored: half-white (resp. half-black). Thus an instance of the problem is a half-colored graph $G = (V, F, B, W, E)$ where F is the set of full vertices (i.e. uncolored yet), B is the set of half-black vertices and W is the set of half-white vertices.

The algorithm is based on a lengthy case analysis (p. 35–49 in [15]) generating reduction and branching rules. The analysis of the running time is based on Eppstein's technique: multivariate recurrences and quasiconvex programming. The recurrences depend on two variables: number of full variables and number of half-colored variables. Once provided the long list of two-variable recurrences they will be solved using Eppstein's quasiconvex programming based approach and one obtains the running time $\mathcal{O}(2^{0.826n})$ of the algorithm.[3]

2.2 A set cover algorithm

A more careful choice of the measure can lead to a significantly better analysis of the worst case running time of simple search tree algorithms. To illustrate this let us consider the following simple exponential-time search tree algorithm for the minimum set cover problem that has been presented in [32] by the authors of this survey. The analysis is based on a sophisticated choice of the measure. This algorithm is used to obtain the fastest known algorithm for the minimum dominating set problem having running time: $\mathcal{O}(2^{0.610n})$ using polynomial space and $\mathcal{O}(2^{0.598n})$ using exponential space.

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

[3]However to verify the stated running time without having a special program on hands is non-trivial.

contained. For the sake of simplicity, we always assume that $\mathcal{S}$ covers $\mathcal{U}$. With this assumption, an instance of MSC is univocally specified by $\mathcal{S}$.

The NP-hard problem Minimum Dominating Set (MDS) asks to determine the smallest cardinality of a dominating set for the input graph G. Recall that a set $D \subseteq V(G)$ is called a dominating set of the graph G if every vertex of G is either in D, or adjacent to some vertex in D. MDS for an input graph G can be naturally reduced to MSC by imposing $\mathcal{U} = V(G)$ and $\mathcal{S} = \{N[v] | \ v \in V\}$, where $N[v]$ denotes the closed neighborhood of vertex v in G. Thus D is a minimum dominating set of G if and only if $\mathcal{S}' = \{N[v] | \ v \in D\}$ is a minimum set cover of $(\mathcal{U}, \mathcal{S})$. Thus an $\mathcal{O}(2^{\alpha(|\mathcal{S}|+|\mathcal{U}|)})$ algorithm for MSC implies an $\mathcal{O}(2^{2\alpha n})$ algorithm for MDS.

Consider the following simple recursive search tree algorithm `msc` for solving MSC:

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

Here $del(S, \mathcal{S}) = \{Z | Z = R \backslash S \neq \emptyset, R \in \mathcal{S}\}$ is the instance of `MSC` which is obtained from $\mathcal{S}$ by removing the elements of S from the subsets in $\mathcal{S}$, and by eventually removing the empty sets obtained. Algorithm `poly-msc` is the polynomial-time minimum set cover algorithm solving MSC for instances where all subsets have cardinality two, which can be reduced to a minimum edge cover problem, based on a well-known reduction to maximum matching.

Essentially algorithm `msc` has two reduction rules (in line 3 and 4) and one branching rule (in line 7). If the maximum cardinality of a subset is at least 3 then the algorithm chooses a subset S of maximum cardinality and branches into the two subproblems $S_{IN} = del(S, \mathcal{S})$ (the case where S belongs to the minimum set cover) and $S_{OUT} = \mathcal{S} \backslash S$ (corresponding to the case S is not in the minimum set cover). It is easy to see that the simple algorithm is correct.

2.2.1 Analyzing the algorithm `msc`

How should we analyze the running time of `msc`? Classical analysis with the natural measure $s(\mathcal{U}', \mathcal{S}') = |\mathcal{S}'| + |\mathcal{U}'|$ for the size of an instance $(\mathcal{U}', \mathcal{S}')$ of MSC provides an upper bound of $\mathcal{O}(2^{0.465(|\mathcal{S}|+|\mathcal{U}|)})$ [40]. (The recurrence corresponding to the unique branching rule is $P(s) \leq P(s-1) + P(s-4)$ where $P(s)$ denotes the number of leaves in the search tree generated by the algorithm to solve a problem of size $s = s(\mathcal{U}, \mathcal{S})$.)

We show how to refine the running time analysis to $\mathcal{O}(2^{0.305(|\mathcal{S}|+|\mathcal{U}|)})$ via a more careful choice of the measure of an instance of MSC (without modifying the algorithm!).

Intuition. The choice is based on the following observations showing two "side effects" not taken into account by the above classical analysis: Removing a large set has a different impact on the "progress" of the algorithm than removing a small one. In fact, when we remove a large set, we decrease the frequency of many elements. Decreasing elements frequency pays of on long term, since the elements of frequency one can be filtered out (without branching). A dual argument holds for the elements. Removing an element of high frequency is somehow preferable to removing an element of small frequency. In fact, when we remove an element occurring in many sets, we decrease the cardinality of all such sets by one. This is good on long term, since sets of cardinality one can be filtered out. This suggests the idea to give a different weight to sets of different cardinality and to elements of different frequency in the measure of an instance.

The measure. Let n_i denote the number of subsets $S \in \mathcal{S}$ of cardinality i. Let moreover m_j denote the number of elements $u \in \mathcal{U}$ of frequency j. The measure $s = s(\mathcal{U}, \mathcal{S})$ of the size of an instance of MSC is defined to be: $s(\mathcal{U}, \mathcal{S}) = \sum_{i \geq 1} w_i \, n_i + \sum_{j \geq 1} v_j \, m_j$, where the weights $w_i, v_j \in (0, 1]$ will be fixed in the following. Note that $s \leq |\mathcal{S}| + |\mathcal{U}|$. Thus when obtaining a running time $\mathcal{O}(2^{\alpha s})$ we may conclude that `msc` has running time $\mathcal{O}(2^{\alpha(|\mathcal{S}|+|\mathcal{U}|)})$.

Notation.

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

Intuitively, $\Delta\, w_i$ ($\Delta\, v_i$) 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 $i-1$. Let us note that this holds also in the case $i = 2$.

Constraints. In order to simplify the running time analysis, we will add the following constraints:

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

Let us observe that this implies that only the weights v_2, v_3, v_4, v_5 and w_2, w_3, w_4, w_5 have still to be fixed. Furthermore for every $i \geq 3$, $w_i \geq w_{i-1}$, and $v_i \geq v_{i-1}$.

Recurrences. Let $P_h(s)$ be the number of subproblems of size h, $0 \leq h \leq s$, solved by `msc` to solve an instance of the MSC of size s. As in a classical analysis for all reduction rules and all branching rules we obtain recurrences. Typically the analysis is more difficult and more tedious than in the case of simple measures because now one branching rule can generate a lot of recurrences.

For the detailed analysis we refer to [32]. We only mention all recurrences corresponding to the unique branching rule (which are practically all important recurrences). Suppose the algorithm has chosen a set S with $|S| \geq 3$ in line 5, thus `msc` branches into two subproblems $\mathcal{S}_{IN} = del(S, \mathcal{S})$ and $\mathcal{S}_{OUT} = \mathcal{S} \backslash S$. Let r_i be the number of elements of S of frequency i. Note that there cannot be elements of frequency 1, and that $\sum_{i \geq 2} r_i = |S|$.

For all the possible values of $|S| \geq 3$ and of the r_i such that $\sum_{i=2}^{6} r_i + r_{\geq 7} = |S|$, we have the following set of recurrences:

$$P_h(s) \leq P_h(s - \Delta\, s_{OUT}) + P_h(s - \Delta\, s_{IN}),$$

where

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

and $\delta(r_2) = 0$ for $r_2 = 0$, and $\delta(r_2) = 1$ otherwise.

Solving recurrences. Fortunately we can restrict our attention to the case $3 \leq |S| \leq 7$. In fact, since $\Delta\, w_{|S|} = 0$ for $|S| \geq 7$, each recurrence with $|S| \geq 8$ is "dominated" by some recurrence with $|S| = 7$.

Thus we consider a large but finite number of recurrences. For every fixed 8-tuple $(w_2, w_3, w_4, w_5, v_2, v_3, v_4, v_5)$ the number $P_h(s)$ is within a polynomial factor of α^{s-h}, where α is the largest number from the set of real roots of the set of equations

$$\alpha^s = \alpha^{s - \Delta\, s_{OUT}} + \alpha^{s - \Delta\, s_{IN}}$$

corresponding to different combinations of values $|S|$ and r_i. Thus the estimation of $P_h(s)$ boils up to choosing the weights minimizing α.

Choosing weights. This optimization problem is interesting in its own and we refer to Eppstein's work [31] on quasiconvex programming for general treatment of such problems. It turns out that $\alpha = \alpha(v, w)$ is a quasiconvex function of the weights (see [31]). We numerically (using a randomized local search algorithm) obtained the following values of the weights:

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

which yields $\alpha \leq 1.2352\ldots < 1.2353$.[4]

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

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

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

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

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

By simply combining the reduction from MDS to MSC with algorithm `msc` one obtains algorithm `mds`.

Corollary 2. *Algorithm* `mds` *solves* MDS *in time* $\mathcal{O}(2^{0.305(2n)}) = \mathcal{O}(2^{0.610n})$.

Applying the memorization technique described in Section 5 to `mds` the running time can be further reduced to $\mathcal{O}(2^{0.598\,n})$.

Though search tree algorithms form a very prominent class of parameterized algorithms, it is yet not fully understood in which way Measure & Conquer can be applied to such algorithms. We leave this as an interesting open problem.

[4] Although computing the weights minimizing α is computationally a non-trivial task, given the weights, checking whether a given α is feasible or not is easy.

3 Exponential lower bounds

Impressive improvements on the upper bound of the worst case running time of a particular exponential-time search tree algorithm can be achieved by a refined analysis and use of a suitable measure as we have seen in the previous section. This suggests the possibility that the time complexity of exponential-time exact algorithms might be largely overestimated. Indeed, most running times of exponential-time search tree algorithms could be too pessimistic and the worst case running time of such an algorithm might be significantly faster. None of the tools and methods for analyzing such algorithms is guaranteed to provide tight upper bounds of the running time.

Consequently, while for most of the known polynomial time algorithms, the known running times seem to be tight, this is most likely not the case for exponential-time search tree algorithms. Therefore it is natural to ask for lower bounds of the worst case running time of such algorithms. A lower bound may give an idea how far the running time analysis is from being tight. Furthermore lower bounds might also help to compare exponential-time search tree algorithms.

There are several results on lower bounds for different so-called DPLL algorithms for SAT and k-SAT (see e.g. [5, 54]). However not much more is known on lower bounds for existing exponential-time search tree algorithms for other problems, and in particular for graph problems. One of the reasons to this could be that for most of the graph problems the construction of good lower bounds seems to be a difficult and challenging task even for very simple algorithms.

3.1 A lower bound for algorithm `mds`

The following lower bound for the $\mathcal{O}(2^{0.610n})$ polynomial-space algorithm `mds` of the previous section has been provided in [32]. Recall that algorithm `mds` solves the Minimum Dominating Set problem based on a reduction to the Minimum Set Cover problem and uses the algorithm `msc`.

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

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

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

We give a selection rule for the choice of the vertices v (respectively sets S_v) to be chosen for the branching. The goal is to choose a selection rule

- which is *compatible* with the algorithm, and
- such that the number of nodes in the search tree obtained by the execution of algorithm `msc` on the instance of MSC generated by the graph G_n is as large as possible.

In each round i, $i \in \{2, 3, \ldots, n-1\}$, we start with a pair $C = \{x_i, y_i\}$ of vertices (belonging to triangle T_i), where $\{x, y\} \subset \{a, b, c\}$. Initially $C = \{a_2, b_2\}$. Our choice makes sure that for each branching vertex x the cardinality of its set S_x is five in the current subproblem $\mathcal{S}$, and that none of the rules of line 2,3 and 4 of the algorithm will ever be applied. Consequently only the branching rule is applied, and by line 7 of `msc` either the set S_v is taken into the set cover ($\mathcal{S} := del(\mathcal{S}, S_v)$), or S_v is removed ($\mathcal{S} := \mathcal{S} \setminus S_v$).

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

The following new pairs of vertices correspond to each of the three branches:
1) $C_1 = \{a_{i+2}, b_{i+2}, c_{i+2}\} \setminus x_{i+2}$,
2) $C_2 = \{a_{i+2}, b_{i+2}, c_{i+2}\} \setminus y_{i+2}$,
3) $C_3 = \{x_{i+1}, y_{i+1}\}$.

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

To show a lower bound on the worst case running time of algorithm `msc` respectively `mds` on input G_n we analyze the number of leaves of the search tree. Let $P(i)$ be the number of leaves in the search tree when all triangles up to T_i have been used for branching. Thus $P(i) = 2 \cdot P(i-2) + P(i-1)$, and hence $P(i) \geq 2^{i-2}$. Consequently the worst case number of leaves in the search tree of `msc` for a graph on n vertices is at least $2^{n/3-2}$. Thus the worst case running time of `mds` is $\Omega(2^{0.333n})$. □

Notice that there is a large gap between the $\mathcal{O}(2^{0.610n})$ upper bound and the $\Omega(2^{0.333n})$ lower bound for the worst case running time of algorithm `mds`. This suggests the possibility that the analysis of algorithm `mds` can be further refined.

4 Tree-width based techniques

The notion of tree-width was introduced by Robertson and Seymour [55]. A *tree decomposition* of a graph G is a pair $(\{X_i : i \in I\},\ T)$, where $\{X_i : i \in I\}$ is a collection of subsets of $V(G)$ and T is a tree such that the following three conditions are satisfied:

1. $\bigcup_{i \in I} X_i = V(G)$.
2. For all $\{v, w\} \in E(G)$, there is an $i \in V(T)$ such that $v, w \in X_i$.
3. For all $i, j, k \in V(T)$, if j is on a path from i to k in T then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree decomposition $(\{X_i : i \in V(T)\},\ T)$ is $\max_{i \in V(T)} |X_i| - 1$. The *tree-width* of a graph G, denoted by $\mathbf{tw}(G)$, is the minimum width over all its tree decompositions. A tree decomposition of G of width $\mathbf{tw}(G)$ is called an *optimal* tree decomposition of G.

A tree decomposition $(\{X_i : i \in V(T)\},\ T)$ of G with T being a path is called a *path decomposition* of G. The *path-width* of a graph G, denoted by $\mathbf{pw}(G)$, is the minimum width over all its path decompositions.

A *branch decomposition* of a graph G is a pair (T, μ), where T is a tree with vertices of degree one or three and μ is a bijection from the set of leaves L of T to $E(G)$. Let e be an edge of T. The removal of e results in two subtrees of T, say T_1 and T_2. Let G_i be the graph formed by the edge set $\{\mu(f) : f \in L \cap V(T_i)\}$ for $i \in \{1, 2\}$. The *middle set* $\mathrm{mid}(e)$ of e is the intersection of the vertex sets of G_1 and G_2, i.e., $\mathrm{mid}(e) := V(G_1) \cap V(G_2)$.

The *width* of (T, μ) is the maximum size of the middle sets over all edges of T, and the *branch-width* of G, $\mathbf{bw}(G)$, is the minimum width over all branch decompositions of G. (In case where $|E(G)| \leq 1$, we define the branch-width to be 0; if $|E(G)| = 0$, then G has no branch decomposition; if $|E(G)| = 1$, then G has a branch decomposition consisting of a tree with one vertex—the width of this branch decomposition is considered to be 0).

Tree-width and branch-width are related parameters and can be considered as measures of the "global connectivity" of a graph. The following result is due to Robertson and Seymour [(5.1) in [56]].

Theorem 4 ([56]). *For any connected graph G with $|E(G)| \geq 3$, $\mathbf{bw}(G) \leq \mathbf{tw}(G) + 1 \leq \frac{3}{2}\mathbf{bw}(G)$.*

Tree-width is one of the most basic parameters in graph algorithms. There is a well established theory on the design of polynomial (or even

linear) time algorithms for many intractable problems when the input is restricted to graphs of bounded tree-width. See [11] for a comprehensive survey. But what is more important for us, many problems on graphs with n vertices and tree-width (branch-width) at most ℓ can be solved in time $c^\ell \cdot n^{\mathcal{O}(1)}$, where c is some problem dependent constant.

For example, Alber et al. [1] proved that MDS on graphs of tree-width at most ℓ can be solved in time $\mathcal{O}(2^{2\ell}n)$. Fomin and Thilikos showed in [36] that for graphs G given with a branch-decomposition of width at most ℓ, a minimum dominating set of G can be computed in time $\mathcal{O}(3^{\frac{3\ell}{2}}m) = \mathcal{O}(2^{5.197n})$. (See also [27] for general discussions on transformations of tree-width based dynamic programming algorithms into algorithms on graphs of bounded branch-width and vice versa.) It can be shown that for graphs of path-width at most ℓ the running time of the algorithm of Alber et al. is $\mathcal{O}(3^\ell n)$.

All results mentioned above are based on the following observation.

Observation 5. *Let $\mathcal{P}$ be a problem on graphs and $\mathcal{G}$ be a class of graphs such that*

- *for every graph $G \in \mathcal{G}$ of branch-width at most ℓ, the problem $\mathcal{P}$ can be solved in time $2^{c_{\mathcal{P}}\ell} \cdot n^{\mathcal{O}(1)}$, where $c_{\mathcal{P}}$ is a constant, and*
- *for every graph $G \in \mathcal{G}$ a branch decomposition (not necessary optimal) of G of width at most $g(n)$ can be constructed in polynomial time.*

Then for every graph $G \in \mathcal{G}$, the problem $\mathcal{P}$ can be solved in time $2^{c_{\mathcal{P}} \cdot g(n)} \cdot n^{\mathcal{O}(1)}$.

Similar observations are valid for tree and path decompositions.

In the following subsections we shall see how Observation 5 combined with good combinatorial upper bounds, provide us with fast algorithms for several interesting graph classes.

4.1 Planar graphs

Using a well-known approach of Lipton and Tarjan [49] based on the celebrated planar separator theorem [48], one can obtain algorithms with time complexity $c^{\mathcal{O}(\sqrt{n})}$ for many problems on planar graphs. However, the constants "hidden" in $\mathcal{O}(\sqrt{n})$ can be crucial for practical implementations. During the last few years some work has been done to compute and to improve the "hidden" constants [3, 4].

Dynamic programming can be seen as a simpler and, sometimes, faster alternative to the approach of Lipton and Tarjan. To use Ob-

servation 5 efficiently, we need to establish upper bounds on the tree-width and branch-width of planar graphs.

Upper bounds. Let α_t and α_b be constants such that for every planar graph $\mathbf{tw}(G) \leq \alpha_t\sqrt{n} + \mathcal{O}(1)$ and $\mathbf{bw}(G) \leq \alpha_b\sqrt{n} + \mathcal{O}(1)$.

In [6] Alon, Seymour, and Thomas proved that any K_r-minor free graph on n vertices has tree-width at most $r^{1.5}\sqrt{n}$. (Here K_r is complete graph on r vertices.) Since no planar graph contains K_5 as a minor, we have that $\alpha_b(G) \leq \alpha_t(G) \leq 6^{1.5} \leq 14.697$. By using deep results of Robertson, Seymour, and Thomas, one can easily prove much better bounds as follows.

Before we proceed, let us remind the notion of a minor. Given an edge $e = \{x, y\}$ of a graph G, the graph G/e is obtained from G by contracting the edge e; that is, to get G/e we identify the vertices x and y and remove all loops and duplicate edges. A graph H obtained by a sequence of edge-contractions is said to be a *contraction* of G. H is a *minor* of G if H is the subgraph of some contraction of G.

The following is a combination of statements (4.3) in [56] and (6.3) in [58].

Theorem 6 ([58]). *Let $k \geq 1$ be an integer. Every planar graph with no $(k \times k)$-grid as a minor has branch-width at most $4k - 3$.*

Since a graph on n vertices does not contain a $((\lceil\sqrt{n}\rceil+1)\times(\lceil\sqrt{n}\rceil+1))$-grid as a minor, we have that $\alpha_b(G) \leq 4$. Fomin and Thilikos [38] obtained the following bounds

Theorem 7 ([38]). $\alpha_b \leq \sqrt{4.5} < 2.1214$ *and* $\alpha_t < 3.1820$.

The proof in [38] makes strong use of deep graph theoretic results from [7] and [57, 62]. In particular, Alon, Seymour and Thomas introduced the concept of "majority" in order to study the existence of small separators in planar graphs. On the other side, the results in [62, 57] are strongly based on the notion of "slope". The main idea of the proof in [38] was to show that slopes can be transformed to majorities.

Now to apply Observation 5, we need to construct a tree or a branch decomposition of small width. It is a long standing open problem whether an optimal tree decomposition of a planar graph can be constructed in polynomial time. For branch decompositions the situation is different. An optimal branch decomposition of a planar graph can be constructed in polynomial time by using the algorithm due to Seymour and Thomas (Sections 7 and 9 in [62]). The algorithm can be implemented such that its running time is $\mathcal{O}(n^4)$. Recently, the running time of the algorithm was reduced by Gu and Tamaki to $\mathcal{O}(n^3)$ [41].

Putting things together. Thus for planar graphs the function $g(n)$ of Observation 5 can be taken $g(n) = \sqrt{4.5n}$. As we already discussed, for Minimum Dominating Set, $c_{\mathcal{P}} \leq 5.1962$, and we arrive at the fastest known algorithms on planar graphs for MDS with running time

$$\mathcal{O}(3^{\frac{3}{2}\sqrt{4.5 \cdot n}}n + n^3) = \mathcal{O}(2^{5.044\sqrt{n}}).$$

Similar approach yields an algorithm for Maximum Independent Set on planar graphs with running time $\mathcal{O}(2^{3.182\sqrt{n}})$.

This machinery not only improves the time bounds but also provides an unified approach for many exponential time algorithms emerging from the planar separator theorem of Lipton and Tarjan [48, 49]. (See [37] for further details.)

Non-local problems. Observation 5 cannot be used to obtain $2^{\mathcal{O}(\sqrt{n})}$ time algorithms on planar graphs for "non-local" problems like Hamiltonian Cycle (HC), where we are asked if the input graph has a Hamiltonian cycle, i.e. a (simple) cycle containing all vertices of the graph. The reason is that all known algorithms, solving HC on graphs of branch-width at most ℓ have running time $2^{\mathcal{O}(\ell \log \ell)}n^{\mathcal{O}(1)}$, thus on planar graphs Observation 5 yields only algorithms with running time $2^{\mathcal{O}(\sqrt{n} \log n)}$.

The intuition, why only $2^{\mathcal{O}(\ell \log \ell)}n^{\mathcal{O}(1)}$ time algorithms for HC on graphs of branch-width at most ℓ are known is the following. While performing dynamic programming, we keep for every edge e of the branch decomposition the set of "patterns" which encode all possible information how possible hamiltonian cycles can hit mid(e). The only known way of doing this is basically to keep as the states of dynamic programming all possible permutations of the set mid(e), which ends up in running time $2^{\mathcal{O}(\ell \log \ell)}n^{\mathcal{O}(1)}$. This seems to be a natural obstacle and no significantly faster algorithm solving Hamiltonian cycle on graphs of bounded branch-width (or tree-width) is known.

Note that for obtaining $2^{\mathcal{O}(\sqrt{n})}$ time algorithms for MDS on planar graphs, planarity comes into play twice: First in the upper bound on the branch-width of a graph and second in the polynomial time algorithm constructing an optimal branch decomposition. It is possible to get rid of the logarithmic factor in the exponent for a number of nonlocal problems as well. The main idea to speed-up algorithms obtained by the branch decomposition approach is to exploit planarity for the third time: use planarity in dynamic programming on graphs of bounded branch-width. To explain how planarity can be used in dynamic programming, we need to go deeper into the properties of planar branch decompositions.

It is more convenient to work with graphs embedded on a sphere instead of a plane. Let Σ be a sphere $(x, y, z\colon x^2 + y^2 + z^2 = 1)$.

By a Σ-*plane* graph G we mean a planar graph G with the vertex set $V(G)$ and the edge set $E(G)$ drawn (without crossing) in Σ. An O-arc is a subset of Σ homeomorphic to a circle. An O-arc in Σ is called *noose* of a Σ-plane graph G if it meets G only in vertices. The *length* of a noose O is $|O \cap V(G)|$, the number of vertices it meets. Every noose O bounds two open discs Δ_1, Δ_2 in Σ, i.e. $\Delta_1 \cap \Delta_2 = \emptyset$ and $\Delta_1 \cup \Delta_2 \cup O = \Sigma$.

For a Σ-plane graph G, we define a *sphere cut branch decomposition* $\langle T, \mu \rangle$ as a branch decomposition such that for every edge e of T there exists a noose O_e bounding the two open discs Δ_1 and Δ_2 such that $G_i \subseteq \Delta_i \cup O_e$, $1 \leq i \leq 2$. Thus the length of the noose O_e is $|\operatorname{mid}(e)|$.

It follows almost directly from results of Seymour and Thomas [62] that the optimal branch decomposition constructed by their algorithm is in fact a sphere cut branch decomposition (see [26] for details).

Let C be a Hamiltonian cycle and let O_e be a noose of a Σ-plane graph G corresponding to an edge e of a sphere cut branch decomposition. Here is the moment when planarity is used for the third time. Because the graph is Σ-plane, the number of possible ways Hamiltonian cycles can hit the noose O_e (which is $\operatorname{mid}(e)$) can be bounded by the $|\operatorname{mid}(e)|$-th Catalan number, which yields almost immediately an algorithm of running time $2^{\mathcal{O}(\mathbf{bw}(G))} n^{\mathcal{O}(1)} = 2^{\mathcal{O}(\sqrt{n})}$.

With a more careful work involving tricks on compressing the number of states in dynamic programming, Dorn et al. [26] established a $\mathcal{O}(2^{6.903\sqrt{n}})$ time algorithm solving HC on planar graphs. A similar approach can be used to obtain an $\mathcal{O}(2^{10.8224\sqrt{n}})$ time algorithm for Planar Graph TSP, where one asks for a shortest tour visiting all vertices of a weighted planar graph. Similarly, Planar Longest Cycle is solvable in time $\mathcal{O}(2^{7.214\sqrt{n}})$.

Finally, let us note that the separator based approach can be used to obtain a $2^{\mathcal{O}(\sqrt{n})}$ time algorithm for HC on planar graphs [23]. However, it seems that by making use of branch decompositions one can prove significantly better bounds on the worst case running time of algorithms on planar graphs.

4.2 Parameterized algorithms on planar graphs

A similar approach (with some modifications) can be used for the design of parameterized algorithms on planar graphs. The last ten years have seen a rapid development of a new branch of computational complexity: Parameterized Complexity. (See the book of Downey and Fellows [28].) Roughly speaking, a parameterized problem with

parameter k is *fixed parameter tractable* if it admits a solving algorithm with running time $f(k)|I|^\beta$. (Here f is a function depending only on k, $|I|$ is the length of the non parameterized part of the input and β is a constant.) In many cases, $f(k) = c^k$ is an exponential function for some constant c. Some attention was paid to the construction of parameterized algorithms with running time of the kind $f(k) = c^{\sqrt{k}}$ for different problems on planar graphs. The first paper on the subject was the paper by Alber et al. [1] describing an algorithm with running time $\mathcal{O}(4^{6\sqrt{34k}}n) = \mathcal{O}(2^{69.972\sqrt{k}}n)$ for the Minimum Dominating Set problem on planar graphs.

Let $\mathcal{L}$ be a parameterized problem, i.e. $\mathcal{L}$ consists of pairs (I, k) where I is the input and k is the *parameter* of the problem. *Reduction to linear problem kernel* is the replacement of problem inputs (I, k) by a reduced problem with inputs (I', k') (linear kernel) with constants c_1, c_2 such that

$$k' \leq c_1 k,\ |I'| \leq c_2 k' \text{ and } (I, k) \in \mathcal{L} \Leftrightarrow (I', k') \in \mathcal{L}.$$

(We refer to Downey and Fellows [28] for discussions on fixed parameter tractability and the ways of constructing kernels.)

Observation 8. *Let $\mathcal{L}$ be a parameterized problem (G, k), where G is a graph such that*

- *there is a linear problem kernel (G', k') computable in time $T_{kernel}(|V(G)|, k)$ with constants c_1, c_2 such that an optimal branch decomposition of G' is computable in time $T_{bw}(|V(G')|)$,*
- *for graphs of branch-width at most ℓ, problem $\mathcal{L}$ can be solved in time $\mathcal{O}(2^{c_3\ell}n)$, where c_3 is a constant, and*
- $\mathbf{bw}(G') \leq c_4\sqrt{k}$*, where c_4 is a constant.*

Then $\mathcal{L}$ can be solved in time $\mathcal{O}(2^{c_3c_4\sqrt{k}}k + T_{bw}(|V(G')|) + T_{kernel}(|V(G)|, k))$.

Proof. The algorithm works as follows. First it computes a linear kernel in time $T_{kernel}(|V(G)|, k)$. Then it constructs a branch decomposition of the kernel G' in time $T_{bw}(|V(G')|)$. (If there is no such kernel, the problems has no solution.) The size of the kernel is at most $c_1c_2k = \mathcal{O}(k)$. The branch-width of the kernel is at most $c_4\sqrt{k}$ and it takes time $\mathcal{O}(2^{c_3c_4\sqrt{k}}k + T_{bw}(|V(G')|) + T_{kernel}(|V(G)|, k))$ to solve the problem. □

Let us on exemplify on parameterize Dominating Set problem how Observation 8 can be used.

The k-Dominating Set problem asks to compute, given a graph G and a positive integer k, a dominating set of size k or to report that no such set exists. Alber, Fellows and Niedermeier [2] show that the k-Dominating Set problem on planar graphs admits a linear problem kernel. (The size of the kernel is $335k$. Recently this result was improved to $67k$ by Chen et al. [17].) This reduction can be performed in $\mathcal{O}(n^3)$ time. As we already mentioned, the MDS on graphs of branch-width at most ℓ can be solved in time $\mathcal{O}(2^{3\log_4 3\cdot\ell}m)$ [36]. Thus $c_3 \leq 3\log_4 3$.

What about the constant c_4 for MDS? It is proved in [36] that for every planar graph G with a dominating set of size k, the branch-width of G is at most $3\sqrt{4.5}\sqrt{k}$, i.e. $c_4 \leq 3\sqrt{4.5}$. Therefore by Observation 8, k- Dominating set can be solved in time $\mathcal{O}(2^{9\cdot\log_4 3\cdot\sqrt{4.5}\sqrt{k}}k + n^3 + k^3) = \mathcal{O}(2^{15.130\sqrt{k}} + n^3)$ on planar graphs. This is the fastest known algorithm for k-Planar Dominating Set.

By similar arguments, one can show that k-Vertex Cover on planar graphs can be solved in time $\mathcal{O}(k^4 + 2^{4.5\sqrt{k}}k + kn)$. (See [37] for details.)

Parameterized versions of non-local problems. For non-local problems Observation 8 cannot be applied directly, however similar arguments are valid. Let us consider the following parameterized version of Hamiltonian Cycle problem: In the k-Cycle problem we are given a graph G and a positive integer k, the task is to find a cycle of length at least k, or to conclude that there is no such a cycle. By adopting the technique from [26], a longest cycle in a planar graph of branch-width at most ℓ can be found in time $\mathcal{O}(2^{3.4\ell}\ell\, n)$. If the branch-width of G is at least $4\sqrt{k+1} - 3$ then by Theorem 6, G contains a $(\sqrt{k+1} \times \sqrt{k+1})$-grid as a minor and thus contains a cycle of length at least k. If the branch-width of G is less than $4\sqrt{k+1} - 3$ then we can find the longest cycle in G in time $\mathcal{O}(2^{3.4\cdot 4\sqrt{k+1}}\sqrt{k}\, n) = \mathcal{O}(2^{13.6\sqrt{k}}\sqrt{k}\, n + n^3)$. By standard techniques (see for example [28]) the recognition algorithm for k-Cycle on planar graphs can easily be turned into one constructing a cycle of length at least k, if such a cycle exists.

The described technique can be applied to a large collection of parameterized problems (so-called bidimensional problems) and it can also be extended to more general graph classes. See [24, 25, 36] for further details.

4.3 Sparse graphs

Another class of graphs for which tree-width based techniques can be used to design exact algorithms are graphs of small maximum degree and graphs with small number of edges.

One of the usual approaches to obtain exact algorithms on sparse graphs are search tree algorithms. There are quite many exact algorithms in the literature for different NP hard problems on sparse graphs and in particular on graphs of maximum degree three, see e.g. [8, 20, 35, 39, 47]

The following result is due to Fomin and Høie [33].

Theorem 9 ([33]). *For any $\varepsilon > 0$, there exists an integer n_ε such that for every graph G with maximum degree at most three and $|V(G)| > n_\varepsilon$, $\mathbf{pw}(G) \leq (1/6 + \varepsilon)|V(G)|$.*

The proof of Theorem 9 provides an algorithm to construct a path decomposition of width at most $(1/6 + \varepsilon)|V(G)|$. Theorem 9 and Observation 5 imply the following

Corollary 10. *For graphs of maximum degree at most three* MDS *is solvable in time $3^{n/6} \cdot n^{\mathcal{O}(1)} = \mathcal{O}(2^{0.265n})$.*

By similar approach one can also obtain the fastest known so far $2^{n/6} \cdot n^{\mathcal{O}(1)} = \mathcal{O}(2^{0.167n})$-time algorithms for Maximum Independent Set and Max-Cut on graphs of maximum vertex degree three. [5]

The proof of Theorem 9 is based on a result of Monien and Preis [50] about the bisection width of 3-regular graphs.

Let us also mention an interesting upper bound on the tree-width of graphs in terms of the number of edges obtained by Kneis et al. [44]

Theorem 11 ([44]). *For any graph G on m edges, $\mathbf{tw}(G) \leq m/5.217$.*

This result implies, for example, that Max-Cut can be solved in time $\mathcal{O}(2^{m/5.217})$.

4.3.1 Lower bounds

The worst case running time of the algorithms described in this subsection depends on combinatorial bounds on path-width of graphs with maximum degree three. Thus it is natural to ask, how small can be

[5] Recently, Kojevnikov and Kulikov [46] announced a new search tree algorithm for Maximum Independent Set on graphs of maximum degree three with running time $2^{n/6} \cdot n^{\mathcal{O}(1)}$.

the path-width or tree-width of graphs of maximum degree three, or even 3-regular graphs.

Lower bounds on these graph parameters can be obtained by making use of Algebraic Graph Theory. In particular, Bezrukov et al. [10] (by making use of the second smallest eigenvalues of Ramanujan graph's Laplacian) showed that there are 3-regular graphs with the bisection width at least $0.082n$. (See [10] for more details.) It can be easily shown that the result of Bezrukov et al. also yields the lower bound $0.082n$ for path-width of graphs with maximum degree three.

The gap between $0.082n$ and $0.167n$ for the upper bound on the path-width of 3-regular graphs provides some hopes for faster algorithms.

5 Memorization

The time complexity of many exponential time search tree algorithms can be reduced at the cost of an exponential space complexity via the *memorization* technique by Robson [59]. Memorization works as follows: the solutions of all the subproblems solved are stored in an (exponential-size) database. 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. The database is implemented in such a way that the *query time* is logarithmic in the number of solutions stored and polynomial in the size of the problem: this way the cost of each look up is polynomial.

In order to illustrate the technique better, we will consider a specific NP-hard problem, the Minimum Vertex Cover problem (MVC), and a specific algorithm to solve it. The techniques described in this section can easily be adapted to many other algorithms and problems. Moreover, for the sake of simplicity, we will analyze the algorithm with the standard measure (using Measure & Conquer, better bounds are achievable).

MVC consists in determining the minimum cardinality of a subset V' of vertices (*vertex cover*) such that every edge is incident to at least one vertex in V'. Let us consider the following simple search tree algorithm to solve MVC: (1) if there is a vertex v of degree zero, remove it; (2) if there is a vertex v of degree one, add w to the vertex cover and remove both v and w (with all the edges incident to them); (3) select v of maximum degree; (3.a) if $deg(v) = 2$, solve the problem with the trivial polynomial-time algorithm; (3.b) otherwise, branch by either including v or its neighborhood $N(v)$ in the vertex cover, and by removing v or its closed neighborhood $N[v]$, respectively. Solve the two subproblems generated recursively. Observe that each subproblem

involves an induced subgraph of the original graph. This property is crucial in order to apply memorization, as it will be clearer soon.[6]

Let $P(n)$ be the number of leaves in the search tree recursively generated by the algorithm to solve the problem on a graph with n vertices. The worst case recurrence, corresponding to the case we branch at a vertex of degree 3, is

$$P(n) \leq P(n-1) + P(n-4),$$

from which we obtain $P(n) < 2^{0.465\,n}$. Since each recursive call takes polynomial time, and the total number of subproblems solved is within a polynomial factor from $P(n)$, the running time of the algorithm (according to the standard analysis) is $\mathcal{O}(2^{0.465\,n})$. Let $P_h(n)$, $h \leq n$, be the number of subproblems being graphs with h vertices solved when the algorithm solves MVC on a graph with n vertices. Observe that, by basically the same analysis, $P_h(n) < 2^{0.465(n-h)}$.

5.1 The basic technique

The running time can be reduced, at the cost of an exponential space complexity, in the following way. Whenever we solve a subproblem G', we store the pair $(G', mvc(G'))$ in a database. Before solving any subproblem, we check whether its solution is already available in the database. Observe that, since G has $\mathcal{O}(2^n)$ induced subgraphs, the database can be easily implemented such that each query takes polynomial time in n.

There are $\binom{n}{h}$ induced subgraphs of G with h vertices, which implies $P_h(n) \leq \binom{n}{h}$ since no subproblem is solved twice. Moreover the upper bound $P_h(n) \leq 2^{0.465(n-h)}$ still holds. Altogether

$$P_h(n) \leq \min\{2^{0.465(n-h)}, \binom{n}{h}\}.$$

By Stirling's approximation, and balancing the two terms, one obtains that, for each h, $P_h(n) \leq 2^{0.465(n-\alpha n)} < 2^{0.425\,n}$, where $\alpha > 0.0865$ satisfies

$$2^{0.465(1-\alpha)} = \frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}.$$

As a consequence, the running time is $\mathcal{O}(2^{0.425\,n})$.

[6] Chen, Kanj and Jia [18] erroneously applied memorization to a MVC algorithm which does not satisfy this property; this mistake was later corrected in the journal version of their paper [19].

5.2 A refined approach

If the graph considered is disconnected, one can solve the vertex cover problem corresponding to each connected component separately. More precisely, if $G_1, G_2, \ldots, G_p$ are the connected components of G, then

$$mvc(G) = \sum_{i=1}^{p} mvc(G_i).$$

This, in combination with memorization, can help to further reduce the running time bound, provided that the degree of the graph is bounded by a small constant. In fact, the number of connected induced subgraphs on h vertices of a graph of maximum degree d is much smaller than $\binom{n}{h}$, provided that h is sufficiently small.

Theorem 12 ([59]). *Let $d \geq 3$ be a constant and G a graph of maximum degree d. Let $G(h)$ be the set of all connected induced subgraphs of G on h vertices. Then*

$$|G(h)| = \mathcal{O}\left(\left(\frac{(d-1)^{d-1}}{(d-2)^{d-2}}\right)^h \; n^{\mathcal{O}(1)}\right).$$

Proof. The claim is trivially true when $h = n$. So let us assume $h < n$. Consider a graph $G' \in G(h)$. Since G is connected, there must be one edge incident to exactly one vertex of G', say $\{u, r\} \in E(G)$ with $r \in V(G')$ and $u \in V - V(G')$.

Let $T'(r)$ be an arbitrary spanning tree of G' rooted at r (there must be one such tree since G' is connected). Consider an arbitrary ordering of the edges. This numbering allows to univocally associate to $T'(r)$ a $(d-1)$-ary tree T'' (where the position of the children of each vertex is taken into account): the neighbors of each vertex w, excluding the parent vertex (u if $w = r$) are ordered following the ordering on the edges; an edge e which is not in $T'(r)$ gives an empty subtree in T'' in the corresponding position.

Thus, given G and the ordering of the edges, there is a one-to-many mapping between $G(h)$ and the set of triples (v, e, T''), where v is a vertex, e is an edge incident to v, and T'' is a $(d-1)$-ary tree. The claim follows by recalling that the number of $(d-1)$-ary trees is upper bounded by $c(d-1)^{d-1}/(d-2)^{d-2}$, for a small constant c [45]. □

For example, if the maximum degree of a graph is at most 4, one obtains

$$P_h(n) = \mathcal{O}(\min\{2^{0.465(n-h)}, (27/4)^h\}),$$

and thus a running time of $\mathcal{O}(2^{0.465(1-\alpha)n}) = \mathcal{O}(2^{0.398\,n})$, where

$$2^{0.465(1-\alpha)} = (27/4)^\alpha \quad \Leftrightarrow \quad \alpha = \frac{\log(2^{0.465})}{\log(2^{0.465}) + \log(27/4)} > 0.1444.$$

This result can easily be extended to the case of arbitrary graphs, by branching on the vertices of degree 5 or larger in a preliminary phase:

$$P(n) \leq \begin{cases} P(n-1) + P(n-6) \\ 2^{0.398\,n} \end{cases} \leq \max\{2^{0.362\,n},\, 2^{0.398\,n}\}.$$

Observe that vertices of degree smaller than two are removed by reduction rules. Thus, without loss of generality, we can consider in the analysis only the connected induced subgraphs of minimum degree 2: even better upper bounds are available on the number of such graphs.

Theorem 13 ([60]). *Let $d \geq 3$ be a constant and G a graph of maximum degree at most d. Let $G(h,2)$ be the set of connected induced subgraphs of G with h vertices and minimum degree at least 2. Then $|G(h,2)| = \mathcal{O}(c(d)^h\, n^{\mathcal{O}(1)})$ where*

$$c(d) = \max_{x \in X} \left\{ 2^{-x_0} \prod_{i=0}^{d-1} \left(\binom{d-1}{i} / x_i \right)^{x_i} \right\},$$

and

$$X = \left\{ x = (x_0, x_1, \ldots, x_{d-1}) \in \mathbb{R}_+^d \mid \sum_{i=0}^{d-1} x_i = 1 \text{ and } \sum_{i=0}^{d-1} i\, x_i = 1 \right\}.$$

Proof. Consider an arbitrary $G' \in G(h,2)$. We consider the same many-to-one mapping from the $(d-1)$-ary trees to the spanning trees of G' as in the proof of Theorem 12, but this time we restrict our attention to the spanning trees with the minimum possible number of leaves ℓ. Note that no two leaves of such spanning trees can be adjacent (otherwise we could create a new spanning tree with one less leaf, which contradicts the minimality assumption). Consider one such tree T' and one of its leaves v. Let $u = u(v)$ be a vertex adjacent to v in G' but not in T', selected arbitrarily. Note that u must exist since the minimum degree is 2, and it must be an internal vertex of T' by the minimality assumption. Let w be the lowest level ancestor of v in T' of degree 3 or larger ($w = r$ is no such vertex exists). We can obtain a different tree T'' with the same number of leaves by adding to T' the edge $e(v) = \{u, v\}$ and by cutting the new cycle introduced at the edge $e'(v) = \{w, w'\}$ right below w in T'. Note that there is a one-to-one mapping between v and both $e(v)$ and $e'(v)$. As a consequence, this replacement of edges can be performed simultaneously on an arbitrary subset of the leaves of the original spanning tree without interference, leading each time to a different spanning tree. This implies that there are at least 2^ℓ distinct spanning trees with ℓ leaves.

Let us give a weight 2^{-h_0} to each spanning tree of G' with $h_0 \geq \ell$ leaves. The weighted sum of such trees is at least one (since there are at least 2^ℓ trees of weight $2^{-\ell}$). As a consequence, the weighted sum of all the spanning trees of the graphs in $G(h,2)$ is an upper bound on $|G(h,2)|$. The number of $(d-1)$-ary trees with h_i vertices of out-degree i, $i \in \{0,1,\ldots,d-1\}$, is upper bounded by

$$\binom{h}{h_0,h_1,\ldots,h_{d-1}} \left(\prod_{i=0}^{d-1} \binom{d-1}{i}^{h_i} \right),$$

where the first factor considers the possible ways to assign out-degrees to vertices, and the second takes into account the positions of the children of each vertex in the tree. Note that the $(h_0,h_1,\ldots,h_{d-1})$ must belong to the following set

$$H = \{(h_0,h_1,\ldots,h_{d-1}) \in \mathbb{N}^d \mid \sum_{i=0}^{d-1} h_i = h \quad \text{and} \quad \sum_{i=0}^{d-1} i\, h_i = h-1\}.$$

With the notation $x_i = h_i/h$ (and letting h tend to infinity),

$$\begin{aligned}
&\sum_H \left(2^{-h_0} \binom{h}{h_0,h_1,\ldots,h_{d-1}} \prod_{i=0}^{d-1} \binom{d-1}{i}^{h_i} \right) \\
&\quad = \mathcal{O}\left(\sum_H \left(2^{-x_0} \prod_{i=0}^{d-1} \left(\binom{d-1}{i} / x_i \right)^{x_i} \right)^h \right) \\
&= \mathcal{O}\left(|H| \left(\max_{x \in X} \left\{ 2^{-x_0} \prod_{i=0}^{d-1} \left(\binom{d-1}{i} / x_i \right)^{x_i} \right\} \right)^h \right).
\end{aligned}$$

The claim follows by observing that, for any constant d, $|H|$ is polynomially bounded. □

For example if the maximum degree is 4, one obtains $|G(h,2)| = \mathcal{O}(5.5981^h)$, corresponding to the case $(x_0,x_1,x_2,x_3) \simeq (0.2440, 0.5359, 0.1962, 0.0239)$. As a consequence, the running time is $\mathcal{O}(2^{0.465(1-\alpha)n}) = \mathcal{O}(2^{0.392\,n})$, where

$$\alpha = \frac{\log(2^{0.465})}{\log(2^{0.465}) + \log(5.5981)} > 0.1576.$$

By the same arguments as above, this running time bound extends to graphs of arbitrary degree. Based on this approach, Robson obtained the currently fastest $\mathcal{O}(2^{0.250\,n})$ exponential space MVC algorithm [60].

Note that the maximization in Theorem 13 must be performed in a very careful way. In fact, underestimating the value of $c(d)$ would lead to wrong running time bounds. The value of $c(d)$ for some values of d are given in Table 1.

Table 1 Upper bounds on $c(d)$ for $d \in \{3, 4, \ldots, 10\}$.

d	$c(d)$
3	3.4143
4	5.5981
5	7.7654
6	9.9275
7	12.0871
8	14.2455
9	16.4031
10	18.5602

5.3 Memorization in parameterized algorithms

The parameterized k-Vertex Cover problem asks to compute, given a graph G and a positive integer k, a vertex cover of size k or to report that no such set exists.

The algorithm described in the previous subsection can be easily adapted to this task: it is sufficient to update k (besides G) at each recursive call in order to keep track of the number of vertices added to the vertex cover along each search path. Using the same notation as in the previous section, but measuring the progress of the algorithm in terms of k (instead of n), we obtain the following tight recurrence

$$P(k) \leq P(k-1) + P(k-3) < 2^{0.552\,k},$$

which corresponds again to the case in which the algorithm branches at a vertex of degree 3. The corresponding running time is $O(2^{0.552\,k})$.

A linear problem kernel of size $2k$ for the k-Vertex Cover problem (not necessary planar) was obtained by Chen et al. [19]. This result is based on graph-theoretical results of Nemhauser and Trotter [51] and Buss and Goldsmith [13]. The running time of the algorithm constructing such a kernel is $\mathcal{O}(kn + k^3)$. Thus $T_{kernel}(|I|, k) = \mathcal{O}(kn + k^3)$.

By applying such a kernalization to each subproblem generated, and using the basic memorization technique described in Section 5.1, one obtains

$$P_h(k) \leq \min\{2^{0.552(k-h)}, \binom{2k}{2h}\}.$$

As a consequence, the running time is $\mathcal{O}(2^{0.552(1-\alpha)k} + kn) =$

$\mathcal{O}(2^{0.528\,k} + kn)$ where $\alpha > 0.044$ satisfies

$$2^{0.552(1-\alpha)} = \left(\frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}\right)^2.$$

By applying a similar (slightly weaker) approach, Niedermeier and Rossmanith [53] derived a $\mathcal{O}(2^{0.360\,k} + kn)$ exponential space vertex cover algorithm from their own $O(2^{0.370\,k} + kn)$ polynomial space algorithm [52].

However, it is not clear a priori how to apply the refined approach of Section 5.2 (based on the number of connected induced subgraphs) to the problem. In fact, consider a vertex cover instance (G, k), where the connected components of G are $G_1, G_2, \ldots, G_p$, with $p \geq 2$. A simple-minded idea is to branch on the subproblems (G_1, k), (G_2, k),..., (G_p, k). Though this approach is correct in principle, it leads to a bad running time bound (since the value of the argument does not decrease in the subproblems).

Chandran and Grandoni [63] described a simple way to circumvent this problem. Suppose the maximum degree is bounded by a constant d. If a connected component contains a small (constant) number of vertices, the corresponding vertex cover problem can be solved in constant time by brute force. Thus, without loss of generality, we can assume that each connected component contains at least $dh + 1$ vertices (and hence at least dh edges), for some constant h to be fixed later. Since each vertex of the vertex cover can cover at most d edges, the size of the minimum vertex cover of each component is at least h. As a consequence, we can branch on the subproblems $(G_i, k-(p-1)h)$ instead of (G_i, k). In fact, if $mvc(G_i) > k - (p-1)h$ for some i, then $mvc(G) > k$. This leads to a new set of recurrences of the kind

$$P(k) \leq \sum_{i=1}^{p} P(k - (p-1)h) \leq 2^{k/h}.$$

By choosing a sufficiently large (but still constant) h, we can ensure that these recurrences are not tight (and thus the worst-case running time is not affected by the branching on the connected components). For example, imposing $h = 3$, one obtains $P(k) < 2^{0.334\,k}$.

By combining this idea with the refined memorization technique described in Section 5.2, one obtains for graphs of degree at most 4 a running time $\mathcal{O}(2^{0.552(1-\alpha)k} + kn) = \mathcal{O}(2^{0.497\,k} + kn)$ where

$$2^{0.552(1-\alpha)} = 5.5981^{2\alpha} \quad \Leftrightarrow \quad \alpha = \frac{\log(2^{0.552})}{\log(2^{0.552}) + 2\log(5.5981)} > 0.0999.$$

Also in this case the same running time bound extends to graphs of arbitrary degree, provided that vertices of degree 5 or larger are

removed in a preliminary phase:

$$P(k) \leq \begin{cases} P(k-1) + P(k-5) \\ 2^{0.552\,k} \end{cases} \leq \max\{2^{0.406\,k}, 2^{0.552\,k}\}.$$

Using this approach, Chandran and Grandoni [63] derived a $\mathcal{O}(2^{0.350\,k} + kn)$ exponential space algorithm from the $\mathcal{O}(2^{0.370\,k} + kn)$ polynomial space algorithm in [52]. This is the currently fastest algorithm for the parameterized k-Vertex Cover problem.[7]

Acknowledgement. Many thanks to Dimitrios M. Thilikos for his helpful remarks and suggestions.

References

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

[2] J. Alber, M. R. Fellows, and R. Niedermeier, *Polynomial-time data reduction for dominating set*, Journal of the ACM, 51 (2004), pp. 363–384.

[3] J. Alber, H. Fernau, and R. Niedermeier, *Graph separators: a parameterized view*, J. Comput. System Sci., 67 (2003), pp. 808–832.

[4] ——, *Parameterized complexity: exponential speed-up for planar graph problems*, J. Algorithms, 52 (2004), pp. 26–56.

[5] M. Alekhnovich, E. Hirsch, and D. Itsykon, *Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas*, in Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2005), vol. 3142 of LNCS, Springer, Berlin, 2004, pp. 84–96.

[6] N. Alon, P. Seymour, and R. Thomas, *A separator theorem for nonplanar graphs*, J. Amer. Math. Soc., 3 (1990), pp. 801–808.

[7] ——, *Planar separators*, SIAM J. Discrete Math., 7 (1994), pp. 184–193.

[8] R. Beigel, *Finding maximum independent sets in sparse and general graphs*, in Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), ACM and SIAM, 1999, pp. 856–857.

[7] Recently Chen et al. [21] announced $\mathcal{O}(1.2740^k + kn) = \mathcal{O}(2^{0.350\,k} + kn)$-time polynomial space algorithm.

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

[10] S. Bezrukov, R. Elsässer, B. Monien, R. Preis, and J.-P. Tillich, *New spectral lower bounds on the bisection width of graphs*, Theoretical Computer Science, 320 (2004), pp. 155–174.

[11] H. L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theoretical Computer Science, 209 (1998), pp. 1–45.

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

[13] J. F. Buss and J. Goldsmith, *Nondeterminism within* P, SIAM J. Comput., 22 (1993), pp. 560–572.

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

[15] J. M. Byskov, *Exact algorithms for graph colouring and exact satisfiability*, PhD thesis, University of Aarhus, Denmark, (August, 2004).

[16] J. M. Byskov, Byskov and D. Eppstein, *An algorithm for enumerating maximal bipartite subgraphs*, manuscript, (2004).

[17] J. Chen, H. Fernau, I. A. Kanj, and G. Xia, *Parametric duality and kernelization: Lower bounds and upper bounds on kernel size*, in Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS 2005), vol. 3403 of LNCS, Springer, Berlin, 2005, pp. 269–280.

[18] J. Chen, I. A. Kanj, and W. Jia, *Vertex cover: further observations and further improvements*, in Proceedings of the 26th Workshop on Graph Theoretic Concepts in Computer Science (WG 1999), vol. 1665 of LNCS, Springer, Berlin, 1999, pp. 313–324.

[19] ——, *Vertex cover: further observations and further improvements*, Journal of Algorithms, 41 (2001), pp. 280–301.

[20] J. Chen, I. A. Kanj, and G. Xia, *Labeled search trees and amortized analysis: improved upper bounds for NP-hard problems*, in Proceedings of the 14th Annual International Symposium on Algorithms and Computation (ISAAC 2003), vol. 2906 of LNCS, Springer, Berlin, 2003, pp. 148–157.

[21] ——, *Simplicity is beaty: Improved upper bounds for vertex cover*, manuscript, 2005.

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

[23] V. G. Deĭneko, B. Klinz, and G. J. Woeginger, *Exact algorithms for the Hamiltonian cycle problem in planar graphs*, Operations Research Letters, (2005), p. to appear.

[24] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos, *Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs*, Journal of the ACM, (2004, to appear).

[25] ——, *Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs*, ACM Trans. Algorithms, 1 (2005), pp. 33–47.

[26] F. Dorn, E. Penninkx, H. Bodlaender, and F. V. Fomin, *Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions*, in Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005), vol. 3669 of LNCS, Springer, Berlin, 2005, pp. 95–106.

[27] F. Dorn and J. A. Telle, *Two birds with one stone: the best of branchwidth and treewidth with one algorithm*, 2005. manuscript, http://www.ii.uib.no/ telle/bib/DT.pdf.

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

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

[30] ——, *The travelling salesman problem for cubic graphs*, in Proceedings of the 8th Workshop on Algorithms and Data Structures (WADS 2003), vol. 2748 of LNCS, Springer, Berlin, 2003, pp. 307–318.

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

[32] F. V. Fomin, F. Grandoni, and D. Kratsch, *Measure and conquer: Domination – a case study*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005), vol. 3580 of LNCS, Springer, Berlin, 2005, pp. 191–203.

[33] F. V. Fomin and K. Høie, *Pathwidth of cubic graphs and exact algorithms*, Technical Report 298, Department of Informatics, University of Bergen, Norway, 2005.

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

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

[36] F. V. Fomin and D. M. Thilikos, *Dominating sets in planar graphs: Branch-width and exponential speed-up*, in 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003), New York, 2003, ACM and SIAM, pp. 168–177.

[37] ——, *A simple and fast approach for solving problems on planar graphs*, in Proceedings of the 21st International Symposium on Theoretical Aspects of Computer Science (STACS 2004), vol. 2996 of LNCS, Springer, Berlin, 2004, pp. 56–67.

[38] ——, *New upper bounds on the decomposability of planar graphs*, Journal of Graph Theory, (2005, to appear).

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

[40] F. Grandoni, *A note on the complexity of minimum dominating set*, Journal of Discrete Algorithms, (to appear).

[41] Q.-P. Gu and H. Tamaki, *Optimal branch-decomposition of planar graphs in $O(n^3)$ time*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005), vol. 3580 of LNCS, Springer, Berlin, 2005, pp. 373–384.

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

[43] K. Iwama, *Worst-case upper bounds for k-SAT*, Bulletin of the EATCS, 82 (2004), pp. 61–71.

[44] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith, *Algorithms based in treewidth of sparse graphs*, in Proceedings of

the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005), LNCS, Springer, Berlin, 2005, to appear.

[45] D. E. Knuth, *The art of computer programming*, Addison-Wesley, second ed., 1975. Vol. 1: Fundamental algorithms.

[46] A. Kojevnikov and A. S. Kulikov, *A new approach for proving upper bounds for MAX-2-SAT*, 2005. manuscript, http://logic.pdmi.ras.ru/ arist/papers.html.

[47] A. S. Kulikov and S. S. Fedin, *Solution of the maximum cut problem in time* $2^{|E|/4}$, Rossiĭskaya Akademiya Nauk. Sankt-Peterburgskoe Otdelenie. Matematicheskiĭ Institut im. V. A. Steklova. Zapiski Nauchnykh Seminarov (POMI), 293 (2002), pp. 129–138, 183.

[48] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[49] ——, *Applications of a planar separator theorem*, SIAM J. Comput., 9 (1980), pp. 615–627.

[50] B. Monien and R. Preis, *Upper bounds on the bisection width of 3- and 4-regular graphs*, in Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001), vol. 2136 of LNCS, Springer, Berlin, 2001, pp. 524–536.

[51] G. L. Nemhauser and L. E. Trotter, Jr., *Properties of vertex packing and independence system polyhedra*, Math. Programming, 6 (1974), pp. 48–61.

[52] R. Niedermeier and P. Rossmanith, *Upper bounds for vertex cover further improved*, in Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science (STACS 1999), vol. 1563 of LNCS, Springer, Berlin, 1999, pp. 561–570.

[53] ——, *On efficient fixed-parameter algorithms for weighted vertex cover*, Journal of Algorithms, 47 (2003), pp. 63–77.

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

[55] N. Robertson and P. D. Seymour, *Graph minors. II. Algorithmic aspects of tree-width*, Journal of Algorithms, 7 (1986), pp. 309–322.

[56] ——, *Graph minors. X. Obstructions to tree-decomposition*, J. Combin. Theory Ser. B, 52 (1991), pp. 153–190.

[57] ——, *Graph minors. XI. Circuits on a surface*, J. Combin. Theory Ser. B, 60 (1994), pp. 72–106.

[58] N. Robertson, P. D. Seymour, and R. Thomas, *Quickly excluding a planar graph*, J. Combin. Theory Ser. B, 62 (1994), pp. 323–348.

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

[60] ——, *Finding a maximum independent set in time* $O(2^{n/4})$, 2001. manuscript, `http://dept-info.labri.fr/ robson/mis/techrep.html`.

[61] U. Schöning, *Algorithmics in exponential time*, in Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS 2005), vol. 3404 of LNCS, Springer, Berlin, 2005, pp. 36–43.

[62] P. D. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241.

[63] L. Sunil Chandran and F. Grandoni, *Refined memorization for vertex cover*, Information Processing Letters, 93 (2005), pp. 125–131.

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

[65] G. Woeginger, *Exact algorithms for NP-hard problems: A survey*, in Combinatorial Optimization - Eureka, you shrink!, vol. 2570 of LNCS, Springer-Verlag, Berlin, 2003, pp. 185–207.

[66] ——, *Space and time complexity of exact algorithms: Some open problems*, in Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC 2004), vol. 3162 of LNCS, Springer-Verlag, Berlin, 2004, pp. 281–290.