Algorithmica (2009) 54: 181–207
DOI 10.1007/s00453-007-9133-3

On Two Techniques of Combining Branching and Treewidth

Fedor V. Fomin · Serge Gaspers · Saket Saurabh · Alexey A. Stepanov

Received: 4 December 2006 / Accepted: 20 November 2007 / Published online: 14 December 2007

Abstract Branch & Reduce and dynamic programming on graphs of bounded treewidth are among the most common and powerful techniques used in the design of moderately exponential time exact algorithms for NP hard problems. In this paper we discuss the efficiency of *simple* algorithms based on combinations of these techniques. The idea behind these algorithms is very natural: If a parameter like the treewidth of a graph is small, algorithms based on dynamic programming perform well. On the other side, if the treewidth is large, then there must be vertices of high degree in the graph, which is good for branching algorithms. We give several examples of possible combinations of branching and programming which provide the fastest known algorithms for a number of NP hard problems. All our algorithms require non-trivial balancing of these two techniques.

In the first approach the algorithm either performs fast branching, or if there is an obstacle for fast branching, this obstacle is used for the construction of a path decomposition of small width for the original graph. Using this approach we give the

A preliminary version of this paper appeared as *Branching and Treewidth Based Exact Algorithms* in the Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC 2006) [15].
Additional support by the Research Council of Norway.

F.V. Fomin · S. Gaspers (✉) · S. Saurabh · A.A. Stepanov
Department of Informatics, University of Bergen, Bergen 5020, Norway
e-mail: serge@ii.uib.no

F.V. Fomin
e-mail: fomin@ii.uib.no

A.A. Stepanov
e-mail: ljosha@ljosha.org

S. Saurabh
The Institute of Mathematical Sciences, Chennai 600 113, India
e-mail: saket@imsc.res.in

fastest known algorithms for MINIMUM MAXIMAL MATCHING and for counting all 3-colorings of a graph.

In the second approach the branching occurs until the algorithm reaches a subproblem with a small number of edges (and here the right choice of the size of subproblems is crucial) and then dynamic programming is applied on these subproblems of small width. We exemplify this approach by giving the fastest known algorithm to count all minimum weighted dominating sets of a graph.

We also discuss how similar techniques can be used to design faster parameterized algorithms.

Keywords Exact exponential time algorithms · Parameterized algorithms · NP hard problems · Treewidth · #3-Coloring · #Minimum dominating set · Minimum maximal matching · k-Weighted vertex cover

1 Introduction

It is a common belief that exponential time algorithms are unavoidable when we want to find an exact solution of a NP hard problem. The last few years have seen an emerging interest in designing exponential time exact algorithms and we recommend recent surveys [11, 20, 28, 31, 32] for an introduction to the topic.

One of the major techniques for constructing fast exponential time algorithms is the Branch & Reduce paradigm. Branch & Reduce algorithms (also called search tree algorithms, Davis-Putnam-style exponential-time backtracking algorithms etc.) recursively solve NP hard combinatorial problems using reduction rules and branching rules. Such an algorithm is applied to a problem instance by recursively calling itself on smaller instances of the problem.

Let us consider a simple example of a branching algorithm: *finding an independent set of maximum size* in a graph $G = (V, E)$. The solution to an instance G of the problem is derived from the solutions of two smaller subproblems $G - \{v\}$ and $G - N[v]$, where v is any given vertex and $N[v]$ is a set of neighbors of v including v. This is because either v does not belong to the solution, or all its neighbors do not. Then the running time $T(n)$ of the algorithm on a graph on n vertices can be bounded by the recurrence $T(n-1) + T(n-|N[v]|)$. The behavior of such an algorithm is poor when the degree of v is small and the usual trick used in branching algorithms is to combine simple branching on vertices of high degree with different branching rules on vertices of small degrees which (usually) leads to a very complicated case analysis. Such a situation is quite typical for many branching algorithms.

Treewidth 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 treewidth (see [3] for a comprehensive survey). What is more important for us here is that many problems on graphs with n vertices and treewidth at most ℓ can be solved in time $\mathcal{O}(c^{\ell} n^{O(1)})$, where c is some problem dependent constant. This observation combined with upper bounds on treewidth was used to obtain fast exponential algorithms for NP hard problems on cubic, sparse and planar graphs [10, 11, 22, 29]. For example, a maximum independent set of a graph given with a tree decomposition of width

at most ℓ can be found in time $\mathcal{O}(2^{\ell}n)$ (see for example [2]). So, a quite natural approach to solve the independent set problem would be to branch on vertices of high degree and if a subproblem with all vertices of small degrees is obtained, then use dynamic programming.

In this paper we show two different approaches based on combinations of branching and treewidth techniques. Both approaches are based on a careful balancing of these two techniques. In the first approach the algorithm either performs fast branching, or if there is an obstacle for fast branching, this obstacle is used for the construction of a path decomposition of small width for the original graph. We call this technique *Branching & Global Application of Width Parameters* and exemplify it on the following problems:

- MINIMUM MAXIMAL MATCHING (MMM): Given a graph G, find a maximal matching of minimum size.
- #3-COLORING: Given a graph G, count the number of 3-colorings of G.

For MMM, a number of exact algorithms can be found in the literature. Randerath and Schiermeyer [27] gave an algorithm of time complexity $\mathcal{O}(1.4422^m)$ for MMM, where m is the number of edges. Raman et al. [26] improved the running time by giving an algorithm of time complexity $\mathcal{O}(1.4422^n)$ for MMM, where n is the number of vertices. Here, using a combination of branching, dynamic programming over bounded treewidth and enumeration of minimal vertex covers we give an $\mathcal{O}(1.4082^n)$ algorithm for MMM. #3-COLORING is a special case of the more general problem $(d, 2)$-CONSTRAINT SATISFACTION PROBLEM ($(d, 2)$-CSP). A systematic study of exact algorithms for $(d, 2)$-CSP was initiated in [1] where an algorithm with running time $\mathcal{O}(1.788^n)$ was given for #3-COLORING. In recent years, the algorithms for $(d, 2)$-CSP have been significantly improved. Notable contributions include papers by Williams [30] and Fürer and Kasiviswanathan [16]. The current fastest algorithm for #3-COLORING has running time $\mathcal{O}(1.770^n)$ [16]. Here we improve this algorithm with our technique of combining branching and treewidth and give an $\mathcal{O}(1.6308^n)$ time algorithm for the problem. This immediately improves the running time of algorithms for counting k-colorings as a consequence.

In the second approach the branching occurs until the algorithm reaches a subproblem with a small number of edges (and here the right choice of the size of subproblems is crucial) and then dynamic programming on bounded treewidth or pathwidth is applied on these subproblems. We term this technique *Branching & Local Application of Width Parameters* and exemplify it on the following problem:

- #MINIMUM WEIGHTED DOMINATING SET (#MWDS): Given a weighted graph G, count the number of dominating sets in G of minimum weight.

Exact algorithms to find a minimum dominating set have attracted a lot of attention. Recently, several groups of authors independently obtained exact algorithms that solve the MINIMUM DOMINATING SET problem in a graph on n vertices faster than the trivial $\mathcal{O}(2^n)$ time brute force algorithm [14, 19, 27]. The fastest known algorithm computes a minimum dominating set of a graph in time $\mathcal{O}(1.5137^n)$ [12]. The algorithm from [12] cannot be used to compute a dominating set of minimum weight (in a weighted graph), however, as it was observed in [13], the weighted version of

the problem can be solved in time $\mathcal{O}(1.5780^n)$ by similar techniques. In the same paper it was shown that all minimal dominating sets can be listed in time $\mathcal{O}(1.7697^n)$ (later improved to $\mathcal{O}(1.7170^n)$), which implies that minimum dominating sets can be counted in this time. Here we give an algorithm that counts minimum weighted dominating sets in a weighted graph on n vertices in time $\mathcal{O}(1.5535^n)$.

Finally we apply our technique to Parameterized Complexity. For decision problems with input size n, and a parameter k (which typically is the solution size), the goal here is to design an algorithm with runtime $f(k)n^{\mathcal{O}(1)}$ where f is a function depending only on k. Problems having such an algorithm are said to be fixed parameter tractable (FPT). The book by Downey and Fellows [7] provides a good introduction to the topic of parameterized complexity. Here, we apply our technique to parameterized k-WEIGHTED VERTEX COVER. The k-VERTEX COVER problem, asking whether an input graph has at most k vertices that are incident to all its edges, is a celebrated example of a FPT problem. When parameterized by k, this problem can be solved in time $\mathcal{O}(n^3 + c^k)$, where c is a constant. After a long race of improvements (see for example [6, 24]), the current best algorithm by Chandran and Grandoni has running time $\mathcal{O}(1.2745^k k^4 + kn)$ [5].

For k-WEIGHTED VERTEX COVER, also known as REAL VERTEX COVER, Niedermeier and Rossmanith [25] gave two algorithms, one with running time $\mathcal{O}(1.3954^k + kn)$ and polynomial space and the other one using time $\mathcal{O}(1.3788^k + kn)$ and space $\mathcal{O}(1.3630^k n^{O(1)})$. Their dedicated paper on k-WEIGHTED VERTEX COVER is based on branching, kernelization and the idea of memorization. Their analysis involves extensive case distinctions when the maximum degree of the reduced graph becomes 3. Here, we give a very simple algorithm running in time $\mathcal{O}(1.3803^k n)$ and space $\mathcal{O}(1.2599^k + kn)$. The other problems for which we give parameterized algorithms include parameterized edge dominating set and its variants.

While the basic idea of our algorithms looks quite natural, the approach is generic and the right application of our approach improves many known results. The novel and most difficult part of the paper is the analysis of the algorithms based on our technique. The running time depends on a careful balancing of different parameters involved in the algorithm. For MMM and #3-COLORING, we carefully count subproblems of different size arising during the recurrence to get the overall improvement. To analyze the running time for #MWDS we investigate the behavior of the pathwidth of a graph as a function of the measure of the corresponding set cover instance. The difficulty here is to find the measure of the problem that "balances" the branching and dynamic programming parts of the algorithm. To choose the right measure we express the running time of the dynamic programming part as a linear program and use the solution of this linear programming formulation to balance different parts of the algorithm. Finally for parameterized algorithms we make use of known "kernels" for the problems and the fact that graphs with maximum degree at most 2 have constant pathwidth.

We use standard dynamic programming algorithms on graphs of bounded pathwidth or treewidth in all of our algorithms. But to use these algorithms it is important that we have good upper bounds on the pathwidth of the subgraphs arising in our recursive algorithms. We prove several upper bounds on the pathwidth of sparse graphs that we use in our algorithms. These bounds are of independent interest.

The rest of the paper is organized as follows. In Sect. 2 we give some basic definitions and notations we use in the paper. We develop some nontrivial upper bounds on pathwidth of sparse graphs in Sect. 3. In Sect. 4, we exemplify the technique of *Branching & Global Application of Width Parameters* on MMM and #3-COLORING. In Sect. 5 we give an algorithm for MWDS as an application of *Branching & Local Application of Width Parameters* technique. We apply our techniques to parameterized algorithms in Sect. 6. Finally, we conclude with some remarks and open problems in Sect. 7.

2 Preliminaries

In this paper we consider simple undirected graphs. Let $G = (V, E)$ be a graph and let n denote the number of vertices and m the number of edges of G. We denote by $\Delta(G)$ the maximum vertex degree in G. For a subset $V' \subseteq V$, $G[V']$ is the graph induced by V', and $G - V' = G[V \setminus V']$. For a vertex $v \in V$ we denote the set of its neighbors by $N(v)$ and its *closed neighborhood* by $N[v] = N(v) \cup \{v\}$. Similarly, for a subset $D \subseteq V$, we define $N[D] = \bigcup_{v \in D} N[v]$. An *independent set* in G is a subset of pair-wise non-adjacent vertices. A subset of vertices $S \subseteq V$ is a *vertex cover* in G if for every edge e of G at least one endpoint of e is in S.

Major tools of our paper are tree and path decompositions of graphs. A *tree decomposition* of G is a pair $(\{X_i : i \in I\}, T)$ where each X_i, $i \in I$, is a subset of V, called a *bag* and T is a tree with elements of I as nodes such that we have the following properties:

1. $\bigcup_{i \in I} X_i = V$;
2. for all $\{u, v\} \in E$, there exists $i \in I$ such that $\{u, v\} \subseteq X_i$;
3. for all $i, j, k \in I$, if j is on the path from i to k in T then $X_i \cap X_k \subseteq X_j$.

The width of a tree decomposition is equal to $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all its tree decompositions and it is denoted by $\mathbf{tw}(G)$. We speak of a *path decomposition* when the tree T in the definition of a tree decomposition is restricted to be a path. The *pathwidth* of G is defined similarly to its *treewidth* and is denoted by $\mathbf{pw}(G)$.

Our $\mathcal{O}^*$ notation suppresses polynomial terms. Thus we write $\mathcal{O}^*(T(x))$ for a time complexity of the form $\mathcal{O}(T(x) \cdot \text{poly}(|x|))$ where $T(x)$ grows exponentially with $|x|$, the input size.

3 Upper Bounds on Pathwidth of Sparse Graphs

In this section we develop several upper bounds on the pathwidth of sparse graph. We need the following known bound on the pathwidth of graphs with maximum degree 3 to prove the two lemmas of this section.

Proposition 1 [10] *For any $\varepsilon > 0$, there exists an integer n_ε such that for every graph G with $n > n_\varepsilon$ vertices and maximum degree at most 3, $\mathbf{pw}(G) \leq (1/6 + \varepsilon)n$. Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.*

Using Proposition 1 we prove the following bound for general graphs.

Lemma 1 *For any $\varepsilon > 0$, there exists an integer n_ε such that for every graph G with $n > n_\varepsilon$ vertices,*

$$\mathbf{pw}(G) \leq \frac{1}{6}n_3 + \frac{1}{3}n_4 + \frac{13}{30}n_5 + \frac{23}{45}n_6 + n_{\geq 7} + \varepsilon n,$$

where n_i is the number of vertices of degree i in G for any $i \in \{3, \ldots, 6\}$ and $n_{\geq 7}$ is the number of vertices of degree at least 7. Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.

Proof Let $G = (V, E)$ be a graph on n vertices. It is well known (see for example [3]) that if the treewidth of a graph is at least 2, then contracting edges adjacent to vertices of degree 1 and 2 does not change the treewidth of a graph and thus increases its pathwidth by at most a logarithmic factor. So we assume that G has no vertices of degree 1 and 2 (otherwise we contract the corresponding edges).

First, we prove the lemma for the special case where the maximum degree of G is at most 4 by induction on the number n_4 of vertices of degree 4 in G. If $n_4 = 0$, then $\Delta(G) \leq 3$, and we apply Proposition 1. Let us assume that for $n_4 \geq 1$ and for every $\varepsilon > 0$ there exists n_ε such that for every graph with at least n_ε vertices and at most $n_4 - 1$ vertices of degree 4 the lemma holds. Let $v \in V$ be a vertex of degree 4. Every neighbor of v has degree at least 3. Let $i \in \{0, \ldots, 4\}$ be the number of degree 3 neighbors of v. Thus v has $4 - i$ neighbors of degree 4 and

$$\begin{aligned}\mathbf{pw}(G) &\leq \mathbf{pw}(G - v) + 1\\ &\leq \frac{n_3 - i + (4 - i)}{6} + \frac{n_4 - 1 - (4 - i)}{3} + \varepsilon(n - 1) + 1\\ &\leq \frac{n_3}{6} + \frac{n_4}{3} + \varepsilon n.\end{aligned}$$

Now, suppose that the maximum degree of G is at most 5. We have already proved the base case where $n_5 = 0$. Let us assume that for some $n_5 \geq 1$ the statement of the lemma holds for all graphs with at most $n_5 - 1$ vertices of degree 5, no vertices of degree at least 6 and at least one vertex of degree at most 4. (The case when the graph is 5-regular requires special consideration.)

Let v be a vertex of degree 5. Let us first assume that the graph $G - v$ is not 5-regular. It is clear that $\mathbf{pw}(G) \leq \mathbf{pw}(G - v) + 1$. For $j \in \{3, \ldots, 5\}$ we denote by m_j the number of degree j neighbors of v. By the induction assumption,

$$\begin{aligned}\mathbf{pw}(G) &\leq \mathbf{pw}(G - v) + 1\\ &\leq \frac{n_3 - m_3 + m_4}{6} + \frac{n_4 - m_4 + m_5}{3} + \frac{13}{30}(n_5 - 1 - m_5) + 1 + \varepsilon n.\end{aligned}$$

For all possible values of $m = (m_3, m_4, m_5)$, we have that

$$\frac{13}{30} \leq \frac{1 + \frac{1}{6}(m_4 - m_3) + \frac{1}{3}(m_5 - m_4)}{1 + m_5}.$$

(The equality is obtained when $m = (m_3, m_4, m_5) = (0, 1, 4)$ which corresponds to the case when v has four neighbors of degree 5 and one of degree 4.) Thus,

$$\mathbf{pw}(G) \leq \frac{n_3}{6} + \frac{n_4}{3} + \frac{13}{30}n_5 + \varepsilon n.$$

If the graph obtained from $G - v$ by contracting edges adjacent to vertices of degree 1 and 2 is 5-regular, then all neighbors of v in G are of degree 3. Let u be a vertex of degree 5 in $G - v$. Since $G - u - v$ is not 5-regular and $\mathbf{pw}(G) \leq \mathbf{pw}(G - u - v) + 2$, we have that

$$\begin{aligned}\mathbf{pw}(G) &\leq \mathbf{pw}(G - u - v) + 2 \\ &\leq 2 + \frac{n_3 - 5}{6} + \frac{n_4 + 5}{3} + \frac{13}{30}(n_5 - 7) + \varepsilon n < \frac{n_3}{6} + \frac{n_4}{3} + \frac{13}{30}n_5 + \varepsilon n.\end{aligned}$$

Thus the lemma holds for all non 5-regular graphs. Since the removal of one vertex changes the pathwidth by an additive factor of at most 1, for sufficiently large n this additive factor is dominated by εn, and we conclude that the lemma holds for 5-regular graphs as well.

Using similar arguments one can proceed with the vertices of degree 6 (we skip the proof here). The critical case here is when the vertex of degree 6 has 5 neighbors of degree 6 and one neighbor of degree 5.

For vertices of degree at least 7 we just use the fact that adding a vertex to a graph can increase its treewidth by at most one. □

The following result bounds treewidth in terms of both the number of vertices and the number of edges and is very useful when we have information about the average degree rather than the maximum degree of the graph.

Lemma 2 *For any $\varepsilon > 0$, there exists an integer n_ε such that for every connected graph G with $n > n_\varepsilon$ vertices and $m = \beta n$ edges, $\beta \in [1.5, 2]$, the treewidth of G is at most $(m - n)/3 + \varepsilon n$. Moreover, a tree decomposition of the corresponding width can be constructed in polynomial time.*

Proof First we show the result assuming that the maximum degree $\Delta(G)$ of the graph is bounded by 4 we then extend this result without any degree constraint.

Let n_3 be the number of vertices of degree 3 in G and n_4 be the number of vertices of degree 4 in G. Since the contraction of an edge adjacent to a vertex of degree one and two does not change the treewidth of a graph, we assume that $n_3 = n - n_4$. Thus $\frac{3}{2}n_3 + 2n_4 = \beta n$. Since $n_4 = (2\beta - 3)n$ and $n_3 = (4 - 2\beta)n$, by Lemma 1 we have that

$$\begin{aligned}\mathbf{tw}(G) \leq \mathbf{pw}(G) &\leq \frac{1}{3}(2\beta - 3)n + \frac{1}{6}(4 - 2\beta)n + \varepsilon n \\ &= \frac{\beta - 1}{3}n + \varepsilon n = \frac{m - n}{3} + \varepsilon n.\end{aligned}$$

Now we extend the result without any assumptions on the degrees of the vertices of G. We show this by induction on $n_{\geq 5}$, the number of vertices of degree at least 5.

We have already shown that the lemma holds if $n_{\geq 5} = 0$. Let us assume that for $n_{\geq 5} \geq 1$, for every $\varepsilon > 0$ there exist n_ε such that for every graph with at least n_ε vertices and at most $n_{\geq 5} - 1$ vertices of degree at least 5 the lemma holds. Let $v \in V$ be a vertex of degree at least 5. Observe that $G - v$ has at most $m - 5$ edges and that $m - 5 \leq 2(n - 1)$. Now we have

$$\begin{aligned}\mathbf{tw}(G) \leq \mathbf{tw}(G - v) + 1 &\leq \frac{m - 5 - (n - 1)}{3} + 1 + \varepsilon n \\ &\leq \frac{m - n}{3} + \varepsilon n = \frac{(\beta - 1)n}{3} + \varepsilon n.\end{aligned}$$

□

4 Branching and Global Application of Width Parameters

In this section we give the fastest known exact algorithms for MMM, its variants and #3-COLORING. Our algorithms either branch on a vertex or compute a path decomposition of the original graph. Once it computes a path decomposition, it stops branching and finds the solution of the problem by applying an algorithm based on dynamic programming on graphs of bounded pathwidth/treewidth for the problem.

4.1 Minimum Maximal Matching

Given a graph $G = (V, E)$, any set of pairwise disjoint edges is called a *matching* of G. A matching M is maximal if there is no matching M' such that $M \subset M'$. The problem of finding a maximum matching is well studied in algorithms and combinatorial optimization. One can find a matching of maximum size in polynomial time but there are many versions of matching which are NP hard. Here, we give an exact algorithm for one such version [17]. More precisely, the problem we study is:

- MINIMUM MAXIMAL MATCHING (MMM): Given a graph $G = (V, E)$ find a *maximal* matching of *minimum* cardinality.

We need the following proposition which is a combination of two classical results due to Moon and Moser [23] and Johnson, Yannakakis and Papadimitriou [21].

Proposition 2 [21, 23] *Every graph on n vertices contains at most $3^{n/3} = \mathcal{O}(1.4423^n)$ maximal (with respect to inclusion) independent sets. Moreover, all these maximal independent sets can be enumerated with polynomial delay.*

The VERTEX COVER problem is the complement of the INDEPENDENT SET problem, that is, $S \subseteq V$ is a vertex cover of G if and only if $V \setminus S$ is an independent set of G and hence Proposition 2 can be used for enumerating minimal vertex covers as well. Our algorithm also uses the following characterization of a MMM.

Proposition 3 [26] *Let $G = (V, E)$ be a graph and M be a minimum maximal matching of G. Let*

$$V[M] = \{v \mid v \in V \text{and } v \text{ is an end point of some edge of } M\}$$

Algorithm `findMMM`(G, H, C)
Input: A graph G, an induced subgraph H of G and a set of vertices $C \subseteq V(G) - V(H)$.
Output: A minimum maximal matching M such that C is contained in the set of end points of the edges of M **or** a path decomposition of G.

```
1  if (Δ(H) ≥ 4) or (Δ(H) = 3 and |C| > 0.17385|V(G)|) then
2      choose a vertex v ∈ V(H) of maximum degree
3      M1 ← findMMM(G, H − N[v], C ∪ N(v))                    (R1)
4      M2 ← findMMM(G, H − {v}, C ∪ {v})                      (R2)
5      return the set of minimum size among {M1, M2}
6  else if (Δ(H) = 3 and |C| ≤ 0.17385|V(G)|) or (Δ(H) ≤ 2 and |C| ≤ 0.31154|V(G)|) then
7      output a path decomposition of G using Lemma 4
8      The algorithm stops.
9  else
10     X ← E(G)
11     foreach minimal vertex cover Q of H do
12         M' ← a maximum matching of G[C ∪ Q]
13         Let V[M'] be the set of end points of M'
14         M'' ← a maximum matching of G[C ∪ V(H) \ V[M']]
15         if M' ∪ M'' is a maximal matching of G and |X| > |M' ∪ M''| then
16             X ← M' ∪ M''
17     return X
```

Fig. 1 Algorithm `findMMM`(G, H, C)

be a subset of all endpoints of M. Let $S \subseteq V[M]$ be a vertex cover of G. Let M' be a maximum matching in $G[S]$ and M'' be a maximum matching in $G - V[M']$, where $V[M']$ is the set of the endpoints of edges of M'. Then $X = M' \cup M''$ is a minimum maximal matching of G.

Note that in Proposition 3, S does not need to be a *minimal* vertex cover.

Finally, we give a lemma for finding a minimum maximal matching on graphs of bounded treewidth, which we use as a subroutine in our algorithm. The proof of the lemma is based on standard dynamic programming on graphs of bounded treewidth and we omit it here.

Lemma 3 *There exists an algorithm to compute a minimum maximal matching of a graph G in time $\mathcal{O}(3^p n^{O(1)})$ when a path decomposition of G of width at most p is given.*

Now we are ready to describe our algorithm for MMM which uses Propositions 2, 3 and Lemma 3 as subroutines. Our detailed algorithm is depicted in Fig. 1. The algorithm of Fig. 1 outputs either a path decomposition of the input graph $G = (V_G, E_G)$ (of reasonable width) or a minimum maximal matching of G. The parameter G of the Algorithm `findMMM` always corresponds to the original input graph, $H = (V_H, E_H)$ is a subgraph of G and C is a vertex cover of $G - V_H$. To solve MMM we run `findMMM`$(G, G, \emptyset)$. The algorithm consists of three parts.

Branch (lines 1–5). The algorithm branches on a vertex v of maximum degree and returns the matching of minimum size found in the two subproblems created according to the following rules:

(**R1**) Vertices $N(v)$ are added to the vertex cover C and $N[v]$ is deleted from H;
(**R2**) Vertex v is added to the vertex cover C and v is deleted from H.

Compute path decomposition (lines 6–8). The algorithm outputs a path decomposition using Lemma 4 (discussed later). Then the algorithm stops without backtracking and a minimum maximal matching is found using the pathwidth algorithm of Lemma 3.

Enumerate minimal vertex covers (lines 9–17). The algorithm enumerates all minimal vertex covers of H. For every minimal vertex cover Q of H, $S = C \cup Q$ is a vertex cover of G and the characterization of Proposition 3 is used to find a minimum maximal matching of G.

The conditions under which these different parts of the algorithm are executed have been carefully chosen to optimize the overall running time of the algorithm, including the pathwidth algorithm of Lemma 3. Note that a path decomposition is computed at most once in an execution of the algorithm as it stops right after outputting the path decomposition. Also note that the minimal vertex covers of H are only enumerated in a leaf of the search tree corresponding to the recursive calls of the algorithm.

We define a *branch node* of the search tree of the algorithm to be a recursive call of the algorithm. A branch node is uniquely identified by the triple (G, H, C), that is the parameters of `findMMM`. Now we give a theorem which proves the correctness and the time complexity of the algorithm.

Theorem 1 *A minimum maximal matching of a graph on n vertices can be found in time* $\mathcal{O}(1.4082^n)$.

Proof We first show the correctness of the algorithm and then bound its running time.

Correctness The algorithm either branches on a vertex of degree at least 3, or produces a path decomposition of G, or enumerates minimal vertex covers as subroutines. The correctness of the step where a path decomposition of G is computed follows from Lemma 3 and the correctness of the branching step is obvious.

If minimal vertex covers of H are enumerated, then the algorithm finds a set of edges X by making use of Proposition 3 for all possible sets $S = C \cup Q$, where Q is a minimal vertex cover of H. Consider a subset of vertices $Q' \subseteq V_H$ such that $C \cup Q'$ forms the set of end points of a minimum maximal matching M of G. Observe that Q' is a vertex cover of H. Hence if $Q \subseteq Q'$ is a minimal vertex cover of H then $S = C \cup Q$ is a vertex cover of G and by applying Proposition 3 for S, a minimum sized maximal matching is obtained for G. Hence, all the minimal vertex covers of H are enumerated as possible candidates for Q.

Time Analysis In the rest of the proof we upper bound the running time of this algorithm. For the running time analysis, it is essential to prove a good bound on the width of the path decomposition of G, obtained by the algorithm.

Lemma 4 *Let* $G = (V, E)$ *be the input graph and* (G, H, C) *be a branch node of the search tree of our algorithm then the pathwidth of the graph is bounded by* $\mathbf{pw}(H) + |C|$. *In particular,*

(a) *If* $\Delta(H) \leq 3$, *then* $\mathbf{pw}(G) \leq (\frac{1}{6} + \varepsilon)|V_H| + |C|$ *for any* $\varepsilon > 0$.
(b) *If* $\Delta(H) \leq 2$, *then* $\mathbf{pw}(G) \leq |C| + 2$.

A path decomposition of the corresponding width can be found in polynomial time.

Proof Let $T = V_G \setminus (V_H \cup C)$ be the set of vertices the algorithm removed from H and did not include in C. Note that the set of endpoints of a maximal matching forms a vertex cover of G. Thus when Algorithm `findMMM` decides that a vertex v is not in the vertex cover C, that is it places it in T, all its neighbors are included in C. Hence, for every branch node (G, H, C) of the search tree of the algorithm we have the following (*a*) *T is an independent set*, and (*b*) $N[V_H] \cap T = \emptyset$. Hence the pathwidth of $G[V_H \cup T]$ is equal to $\mathbf{pw}(H)$. Given a path decomposition P of $G[V_H \cup T]$, we obtain a path decomposition P' of G by adding C to every bag of P. The pathwidth of P' is therefore bounded by $\mathbf{pw}(H) + |C|$. The remaining part of the lemma follows from Proposition 1 and the fact that graphs with maximum degree at most 2 have pathwidth at most 2. □

Let us resume the proof of the theorem. Let $\alpha = 0.17385$ and $\beta = 0.31154$. First, consider the conditions under which a path decomposition may be computed. By combining the pathwidth bounds of Lemma 4 and the running time of the algorithm of Lemma 3, we obtain that MMM can be solved in time

$$\mathcal{O}^*(\max(3^{(1+5\alpha)/6}, 3^{\beta})^n)$$

when the path decomposition part of the algorithm is executed.

Assume now that the path decomposition part is not executed. Then, the algorithm continues to branch when the maximum degree $\Delta(H)$ of the graph H is 3. And so, $|C| > \alpha n$ when $\Delta(H)$ first becomes 3. At this point, the set C has been obtained by branching on vertices of degree at least 4 and we investigate the number of subproblems obtained so far. Let L be the set of nodes in the search tree of the algorithm that correspond to subproblems where $\Delta(H)$ first becomes 3. Note that we can express $|L|$ by a two parameter function $A(n, k)$ where $n = |V(H)|$ and k is the minimum number of vertices that have to be added to C to continue branching. Initially, $n = |V(G)|$ and $k = \alpha n$. This function can be upper bounded by a two parameter recurrence relation corresponding to the unique branching rule of the algorithm:

$$A(n, k) = A(n - 1, k - 1) + A(n - 5, k - 4).$$

When the algorithm branches on a vertex v of degree at least 4 the function is called on two subproblems. If v is not added to C (**(R1)**), then $|N[v]| \geq 5$ vertices are removed from H and $|C|$ increases by $|N(v)| \geq 4$. If v is added to C (**(R2)**), then both parameters decrease by 1.

Let r be the number of times the algorithm branches in the case **(R1)**. Observe that $r \in [0, k/4]$. Let L_r be a subset of L such that the algorithm has branched exactly r times according to **(R1)** in the unique paths from the root to the nodes in L_r. Thus, $|L|$ is bounded by $\sum_{i=0}^{k/4} |L_i|$.

To bound the number of nodes in each L_i, let $l \in L_i$ and P_l be the unique path from l to the root in the search tree. Observe that on this path the algorithm has branched

$k-4i$ times according to **(R2)** and i times according to **(R1)**. Hence, the length of the path P_l is $k-3i$. By counting the number of sequences of length $k-3i$ where the algorithm has branched exactly i times according to **(R1)**, we get $|L_i| \leq \binom{k-3i}{i}$. Thus if the path decomposition is not computed, the time complexity $T(n)$ of the algorithm is

$$T(n) = \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}T'(n-5i-(k-4i))\right)$$
$$= \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}T'(n-i-k)\right) \quad (1)$$

where $T'(n')$ is the time complexity to solve a problem on a branch node (G, H, C) in L with $n' = |V_H|$. (Let us remind that in this case the algorithm branches on vertices of degrees 3 and enumerates minimal vertex covers of H.) Let $p = (\beta-\alpha)n$. To bound $T'(n')$, we use similar arguments as before and use Proposition 2 to bound the running time of the enumerative step of the algorithm. Thus we obtain the following.

$$T'(n') = \mathcal{O}^*\left(\sum_{i=0}^{p/3}\binom{p-2i}{i}3^{\frac{n'-4i-(p-3i)}{3}}\right)$$
$$= \mathcal{O}^*\left(3^{(n'-p)/3}\sum_{i=0}^{p/3}\binom{p-2i}{i}3^{-i/3}\right). \quad (2)$$

We bound $T(n')$ by $\mathcal{O}(3^{(n'-p)/3}d^p)$ for some constant $d \in (1,2)$, the value of d will be determined later. Substituting this in (1), we get:

$$T(n) = \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}3^{\frac{n-i-k-p}{3}}d^p\right) = \mathcal{O}^*\left(3^{(1-\beta)n/3}d^p\sum_{i=0}^{k/4}\binom{k-3i}{i}3^{-i/3}\right).$$

Further suppose that $\sum_{i=0}^{k/4}\binom{k-3i}{i}3^{-i/3}$ sums to $\mathcal{O}(c^k)$ for a constant $c \in (1,2)$, then the overall time complexity of the algorithm is bounded by

$$\mathcal{O}^*((3^{(1-\beta)/3}d^{\beta-\alpha}c^{\alpha})^n).$$

Claim *$c < 1.3091$ and $d < 1.3697$.*

Proof The sum over binomial coefficients $\sum_{i=0}^{k/4}\binom{k-3i}{i}3^{-i/3}$ is bounded by $(k/4)B$ where B is the maximum term in this sum. Let as assume that $B = \binom{k-3i}{i}3^{-i/3}$ for some $i \in \{1,2,\ldots,k/4\}$. To compute the constant c, let $r := c-1$. We obtain

$$B = \binom{k-3i}{i}3^{-i/3} \leq \frac{(1+r)^{k-3i}}{r^i}3^{-i/3}.$$

Here we use the well known fact that for any $x > 0$ and $k \in \{0, \ldots, n\}$,

$$\binom{n}{k} \leq \frac{(1+x)^n}{x^k}.$$

By choosing r to be the minimum positive root of $\frac{(1+r)^{-3}}{r} 3^{-1/3} - 1$, we arrive at $B < (1+r)^k$ for $r \in (0.3090, 0.3091)$. Thus $c < 1.3091$. The value of d is computed in a similar way. □

Finally, we get the following running time for Algorithm `findMMM` by substituting the values for α, β, c and d:

$$\mathcal{O}^*(\max(3^{(1-\beta)/3} d^{\beta-\alpha} c^{\alpha}, 3^{(1+5\alpha)/6}, 3^{\beta})^n) = \mathcal{O}(1.4082^n). \quad \square$$

4.2 Some Variations of MMM

In this subsection we give exact algorithms for two problems which are closely related to MINIMUM MAXIMAL MATCHING.

- MINIMUM EDGE DOMINATING SET (MEDS): Given a graph $G = (V, E)$, find a minimum set of edges $D \subseteq E$ such that every edge of G is either in D or is adjacent to an edge in D.
- MATRIX DOMINATION (MD): Given a $m \times n$ matrix M with entries $\{0, 1\}$, find a minimum set of non-zero entries in M that dominates all other, that is, a minimum sized subset $C \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, n\}$ such that $M_{i,j} = 1$ for all $(i, j) \in C$ and whenever $M_{i,j} = 1$, there exists an $(i', j') \in C$ for which either $i = i'$ or $j = j'$.

Our algorithm for MEDS depends on an old result which shows that every minimum maximal matching is a MEDS [18].

Proposition 4 [18] *Let $G = (V, E)$ be a graph. Then every minimum maximal matching of G is a minimum edge dominating set.*

Proposition 4 in connection with Theorem 1 gives us the following corollary.

Corollary 1 *A* MINIMUM EDGE DOMINATING SET *of a graph with n vertices can be found in time $\mathcal{O}(1.4082^n)$.*

MATRIX DOMINATION reduces to finding a MEDS in a bipartite graph [18]. We obtain an improved algorithm for MATRIX DOMINATION by using the fact that all minimal vertex covers of a triangle free graph (and a bipartite graph in particular) on n vertices can be listed in time $\mathcal{O}^*(2^{n/2})$ [4]. The proof of the following theorem is similar to the one of Theorem 1.

Theorem 2 *Given a matrix M of size $m \times n$ with entries in* $\{0, 1\}$, MATRIX DOMINATION *can be solved in time $\mathcal{O}(1.3918^{m+n})$.*

Proof To solve MATRIX DOMINATION we solve MMM on a bipartite graph on $n+m$ vertices. As observed by Byskov [4], all minimal vertex covers of a triangle free graph (and a bipartite graph in particular) on $n+m$ vertices can be listed in time $\mathcal{O}^*(2^{(n+m)/2})$. Thus the minimal vertex covers of the graph H in Algorithm `find-MMM` can be listed faster, and similar to Theorem 1 we can estimate the running time of the algorithm in this case by balancing the running time of the algorithm based on a path decomposition of the graph with

$$\mathcal{O}^*((2^{(1-\beta)/2}d^{\beta-\alpha}c^{\alpha})^{n+m}) \tag{3}$$

where $\mathcal{O}(d^{(\beta-\alpha)(n+m)})$ is solution to $\sum_{i=0}^{p/3}\binom{p-2i}{i}2^{-i/2}$ while $\mathcal{O}(c^{\alpha(n+m)})$ is solution to $\sum_{i=0}^{k/4}\binom{k-3i}{i}2^{-i/2}$. The values we get for the constants are: $\alpha \leq 0.16110$, $\beta \leq 0.30091$, $d \leq 1.3744$, $c \leq 1.3127$. Substituting these values in (3), we obtain the claim of the theorem. □

4.3 Counting 3-Colorings (#3-COLORING)

A *proper coloring* of a graph is an assignment of colors to its vertices such that no edge is monochromatic. Given a graph $G=(V,E)$, COLORING problem asks for a proper coloring of V, minimizing the number of colors used on the vertices. The problem of 2-COLORING, that is, can the given graph be colored with at most 2 colors is polynomial time solvable (bipartite graph testing) but r-COLORING is NP-complete for any $r \geq 3$ [17]. Here we look at the counting version of 3-COLORING which is defined below.

- 3-COLORING: Given a graph $G=(V,E)$ find a function $c : V \to \{R, B, G\}$ such that for every $\{u,v\} \in E$ we have $c(u) \neq c(v)$.

We denote by #3-COLORING the problem to count all 3-colorings of a graph. Our algorithm for #3-COLORING is very similar to the one presented for MMM. Here we give a simple description for the algorithm without going into details.

We associate two colors $\{R, BG\}$, with every vertex and branch on this color set. While branching if we decide to color a vertex v with R then we color its neighbors BG and remove $N[v]$ from the graph else we color v with BG and remove it from the graph. Let C_1 be the set of vertices of G which are colored R and let $C_2 = V \setminus C_1$ be the vertices colored BG. Now G has a 3-coloring respecting this precoloring if and only if $G[C_2]$ is bipartite and the number of 3-colorings respecting this precoloring is the number of 2-colorings of $G[C_2]$ which is 2^t, where t is the number of connected components of $G[C_2]$. Hence, given a fixed precoloring of a graph G with R and BG, we can compute the number of 3-colorings respecting this precoloring in polynomial time.

We also need the following dynamic programming algorithm on graphs with bounded treewidth for our algorithm.

Lemma 5 *Given a graph $G=(V,E)$ with a tree decomposition of G of width ℓ,* #3-COLORING *can be solved in time $\mathcal{O}(3^{\ell}n^{O(1)})$.*

As in the algorithm for MMM we have three phases in the algorithm. Here H is the induced subgraph on uncolored vertices of G at some recursive step in the algorithm.

Branch. The algorithm branches on a vertex v of maximum degree in H and returns the sum of #3-COLORING found in the two subproblems created according to the following rules:

(**R1**) The vertices in $N(v)$ are added to the color class C_2, v is added to the color class C_1 and $N[v]$ is deleted from H;
(**R2**) The vertex v is added to the color class C_2 and v is deleted from H.

Compute path decomposition. If the maximum degree of H is at most 3 and the size of C_2 is at most $0.3342n$ or if the maximum degree of H is at most 2 and the size of C_2 is at most $0.44517n$, then this is a step for applying pathwidth algorithm on the original graph. At this point, the algorithm outputs a path decomposition and the algorithm stops without backtracking. Then #3-COLORING is solved using the pathwidth algorithm of Lemma 5 on the original graph G.

Enumerate 2-colorings of H. When the maximum degree of H is at most 2 and the size of C_2 does not satisfy the conditions of path decomposition phase then the algorithm enumerates all possible two colorings with R and BG of H to get the coloring of whole graph G with R and BG.

Let us observe here that the analysis and the algorithm for #3-COLORING remains the same except the role of C in the algorithm for MMM is taken by C_2 in the algorithm for #3-COLORING. If we replace C with C_2 in Lemma 4 then we get the same upper bounds on the pathwidth of the original graph G. In the algorithm for #3-COLORING we enumerate all proper 2 colorings of H. This is different than enumerating maximal independent sets of H as we did in the algorithm for MMM. This change leads to use of different α and β than in MMM to optimize the running time of the algorithm for #3-COLORING. Let $T(n)$ denote the time taken by the algorithm for #3-COLORING on graphs on n vertices. For a fixed $\alpha \leq 0.3342$, $\beta \leq 0.44517$, we fix $k = \alpha n$ and $p = (\beta - \alpha)n$. Then the running time of the algorithm is bounded by the following sum when pathwidth algorithm is not used on the graph.

$$T(n) = \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}T'(n-5i-(k-4i))\right)$$

$$= \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}T'(n-i-k)\right),$$

and

$$T'(n') = \mathcal{O}^*\left(\sum_{i=0}^{p/3}\binom{p-2i}{i}2^{n'-4i-(p-3i)}\right) = \mathcal{O}^*\left(2^{(n'-p)}\sum_{i=0}^{p/3}\binom{p-2i}{i}2^{-i}\right).$$

Here $T'(n')$ is the running time of the algorithm on H when its maximum degree is 3 and the size of C_2 (vertices colored with BG) is at least $0.3342n$. We bound $T(n')$

by $\mathcal{O}^*(2^{(n'-p)}d^p)$ for some constant $d \in (1,2)$, the value of d will be determined later. Substituting this in the expression for $T(n)$, we get

$$T(n) = \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}2^{n-i-k-p}d^p\right) = \mathcal{O}^*\left(2^{(1-\beta)n}d^p\sum_{i=0}^{k/4}\binom{k-3i}{i}2^{-i}\right).$$

Further suppose that $\sum_{i=0}^{k/4}\binom{k-3i}{i}2^{-i}$ sums to $\mathcal{O}(c^k)$ for a constant $c \in (1,2)$, then the overall time complexity of the algorithm is bounded by

$$\mathcal{O}^*((2^{(1-\beta)}d^{\beta-\alpha}c^{\alpha})^n).$$

Similar to the analysis in the algorithm for MMM, we get values for α, β, c and d. Finally, we get the following running time for the algorithm for #3-COLORING by substituting the values for $\alpha = 0.3342$, $\beta = 0.44517$, $c = 1.2538$ and $d = 1.2972$

$$\mathcal{O}^*(\max(2^{(1-\beta)}d^{\beta-\alpha}c^{\alpha}, 3^{(1+5\alpha)/6}, 3^{\beta})^n) = \mathcal{O}(1.6308^n).$$

This gives us the following theorem.

Theorem 3 *Let $G = (V, E)$ be an undirected graph on n vertices then* #3-COLORING *can be solved in $\mathcal{O}(1.6308^n)$ time.*

This improves on the $\mathcal{O}(1.770^n)$ time algorithm presented in [16]. We can also use this to obtain faster algorithms for counting k-colorings as done in [1].

5 Branching and Local Application of Width Parameters

In this section, we give an algorithm to count all minimum dominating sets of a graph. The recursive part of this algorithm uses a transformation to the SET COVER problem as in [19]. The analysis of the algorithm is largely based on the Measure & Conquer analysis in [12].

5.1 Counting Minimum Dominating Sets

Given a graph $G = (V, E)$ a set $D \subseteq V$ is called a *dominating set* for G if every vertex from V is either in D, or adjacent to some vertex in D. We define the problem of finding a minimum weighted dominating set as follows:

- MINIMUM WEIGHTED DOMINATING SET (MWDS): Given a graph $G = (V, E)$ with a weight function $w : V \to \mathbb{R}^+$, find a dominating set $D \subseteq V$ of minimum weight $w(D) = \sum_{v \in D} w(v)$.

We denote by #MWDS the counting version of MWDS where the objective is to count all dominating sets of minimum weight. We also need the definition of

- MINIMUM WEIGHTED SET COVER (MWSC): Given a set of elements $\mathcal{U}$ and a collection $\mathcal{S}$ of non-empty subsets of $\mathcal{U}$ along with weight function $w : \mathcal{S} \to \mathbb{R}^+$, find a subset $\mathcal{S}^* \subseteq \mathcal{S}$ of minimum weight $w(\mathcal{S}^*) = \sum_{S \in \mathcal{S}^*} w(S)$ which *covers* $\mathcal{U}$; that is,

$$\bigcup_{S \in \mathcal{S}^*} S = \mathcal{U}.$$

We denote by #MWSC the problem of counting all set covers of minimum weight.

The *frequency* of an element $u \in \mathcal{U}$ is the number of sets $S \in \mathcal{S}$ in which u is contained. We denote it by $\text{freq}(u)$.

#MWDS can be reduced to #MWSC by imposing $\mathcal{U} = V$ and $\mathcal{S} = \{N[v] \mid v \in V\}$. Given a weight function $w(v)$ for MWDS we define the corresponding weight for a set $S \in \mathcal{S}$ as follows,

$$w(S) = w(S = \{N[v] \mid v \in V\}) = w(v).$$

Thus D is a dominating set of G if and only if $\{N[v] \mid v \in D\}$ is a set cover of $\{N[v] \mid v \in V\}$.

We also need a reduction from MWSC to a version of MWDS called

- MINIMUM RED-BLUE WEIGHTED DOMINATING SET (RBWDS): Given a bipartite graph $G = (V, E)$ with a bipartition $V = V_{\text{Red}} \cup V_{\text{Blue}}$ and a weight function $w : V \to \mathbb{R}^+$, find $D \subseteq V_{\text{Red}}$ of minimum weight such that every vertex in V_{Blue} is adjacent to a vertex of D.

To an instance $(\mathcal{U}, \mathcal{S}, w)$ of MWSC one can associate an *incidence graph* $G_{\mathcal{S}}$, which is a bipartite graph on the vertex set $\mathcal{S} \cup \mathcal{U}$ with the bipartition $(\mathcal{S}, \mathcal{U})$, and vertices $S \in \mathcal{S}$ and $u \in \mathcal{U}$ are adjacent if and only if u is an element of S. Let us observe, that $\mathcal{S}$ has a cover of weight k if and only if its incidence graph has a red-blue (with $V_{\text{Red}} = \mathcal{S}$ and $V_{\text{Blue}} = \mathcal{U}$) dominating set of weight k.

Now we give an algorithm that counts minimum weight dominating sets in a weighted graph on n vertices in time $\mathcal{O}(1.5535^n)$. The basic idea is as follows. First we turn the instance for #MWDS into an instance for #MWSC (this idea was first used by Grandoni [19]) and perform branching on large sets and sets of size three containing elements of high degree. After the branching step, we turn the instance for #MWSC into an instance for RBWDS on bipartite graphs and use dynamic programming to count its all solutions.

We also need the following result which can be obtained by a standard dynamic programming technique.

Lemma 6 *All minimum red-blue weighted dominating sets of a bipartite graph G with bipartition $(V_{\text{Red}}, V_{\text{Blue}})$ given together with a path decomposition of G of width at most p can be counted in time $\mathcal{O}(2^p n^{O(1)})$.*

We consider a recursive algorithm `countMWSC` for solving #MWSC. The algorithm depends on the following observation.

Lemma 7 *For a given instance of $(\mathcal{S}, w)$, if there is an element $u \in \mathcal{U}(\mathcal{S})$ that belongs to a unique $S \in \mathcal{S}$, then S belongs to every set cover.*

Algorithm `countMWSC`$(\mathcal{S}, w)$
Input: A collection on sets $\mathcal{S}$ and a weight function $w : \mathcal{S} \to \mathbb{R}^+$.
Output: A couple (*weight,num*) where *weight* is the minimum weight of a set cover of $\mathcal{S}$ and *num* is the number of different set covers of this weight.

```
1  if |S| = 0 then
2      return (0,1)
3  if ∃u ∈ U(S) : freq(u) = 1 then
4      Let u ∈ S'
5      return countMWSC(remove(S', S), w)
6  Pick S ∈ S of maximum cardinality
7  if |S| ≤ 3 and for every S ∈ S the degree of all its elements is at most 6 then
8      return countPW(S, w)
9  else
10     (w_in, n_in) ← countMWSC(remove(S, S), w)
11     (w_out, n_out) ← countMWSC(S \ {S}, w)
12     w_in ← w_in + w(S)
13     if w_in < w_out then
14         return (w_in, n_in)
15     else if w_in = w_out then
16         return (w_in, n_in + n_out)
17     else
18         return (w_out, n_out)
```

Fig. 2 Algorithm `countMWSC`$(\mathcal{S}, w)$

Let us go through the Algorithm `countMWSC`; see Fig. 2. First, if $|\mathcal{S}| = 0$ then the size of the minimum weighted set cover is 0 and the number of such set covers is 1. Otherwise (lines 3–5), the algorithm tries to reduce the size of the problem by checking whether the condition of Lemma 7 is applicable. Specifically, if there is an element $u \in S'$ of frequency one, then S' has to be in every minimum set cover. Thus we remove S' and all its elements from the other sets $S \in \mathcal{S}$. Namely `remove`$(S', \mathcal{S}) = \{Z \mid Z = S \setminus S', S \in \mathcal{S}\}$.

If the condition of Lemma 7 does not apply, then we choose a set $S \in \mathcal{S}$ of maximum cardinality. If $|S| \leq 3$ and for every $S \in \mathcal{S}$ the degree of all its elements is at most 6 then we solve the problem with the algorithm `countPW`. We discuss this algorithm and its complexity later.

Otherwise we branch on the following two smaller subproblems. The first subproblem (`remove`$(S, \mathcal{S}), w)$ corresponds to the case where S belongs to the minimum set cover. Whereas in the subproblem $(\mathcal{S} \setminus \{S\}, w)$, S does not belong to the minimum set cover.

Finally we compare the weights of the two subproblems and return the total weight and number (lines 13–18).

The function `countPW` computes a minimum set cover for a specific instance $(\mathcal{S}, w)$. Namely, $\mathcal{S}$ consists of sets with cardinalities at most 3 and for every $S \in \mathcal{S}$ the degree of all its elements is at most 6. For such a set $\mathcal{S}$, the function `countPW` does the following:

– Construct the incidence graph $G_{\mathcal{S}}$ of $\mathcal{S}$;

– Count the number of minimum red-blue dominating sets in $G_{\mathcal{S}}$ (to perform this step, we construct a path decomposition of $G_{\mathcal{S}}$ and run the dynamic programming algorithm described in Lemma 6).

We show that the running time of the algorithm `countMWSC` is $\mathcal{O}(1.2464^{|\mathcal{S}|+|\mathcal{U}|})$. The analysis is based on the *Measure & Conquer* technique [11, 12] combined with a linear programming formulation of the running time of `countPW`. The analysis of the branching part of the algorithm is quite similar to analysis from [12].

Let n_i be the number of subsets $S \in \mathcal{S}$ of cardinality i and let m_j be the number of elements $u \in \mathcal{U}$ of frequency j. We use the following measure $k = k(\mathcal{S})$ of the size of $\mathcal{S}$:

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

where the weights $w_i, v_j \in (0, 1]$ will be fixed later. Note that $k \leq |\mathcal{S}| + |\mathcal{U}|$. Let

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

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 (the frequency of an element) from i to $i-1$. Note that this also holds for Δv_2, because the new element of frequency one is removed before next branching. And thus we get the total reduction $1 - (1 - v_2) = v_2$.

Theorem 4 *Algorithm* `countMWSC` *solves* #MWSC *in time* $\mathcal{O}(1.2464^{|\mathcal{S}|+|\mathcal{U}|})$.

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

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

Note that this implies $w_i \geq w_{i-1}$ for every $i \geq 2$.

Let $P_h(k)$ be the number of subproblems of size $h \in \{0, \ldots, k\}$, solved by `countMWSC` to solve a problem of size k. Clearly, $P_k(k) = 1$. Consider the case $h < k$ (which implies $|\mathcal{S}| \neq 0$). If the condition in line 3 of the algorithm holds, we remove one set from $\mathcal{S}$. Thus the reduction of size of the problem is at least w_1 (worst case, $|S| = 1$) and $P_h(k) \leq P_h(k - w_1)$. Otherwise, let S be the subset selected in line 6. If $|S| \leq 3$ and for every $S \in \mathcal{S}$ the degree of all its elements is at most 6 (line 7), no subproblem is generated. Otherwise, we branch on two subproblems $S_{\text{out}} = (w_{\text{out}}, n_{\text{out}})$ and $S_{\text{in}} = (w_{\text{in}}, n_{\text{in}})$.

Consider subproblem S_{out}. It corresponds to the case where S does not belong to the set cover. The size of S_{out} decreases by $w_{|S|}$ because of the removal of S. Let m_i be the number of elements of S with frequency i. Note that there cannot be elements of frequency 1. Consider an element $u \in S$ with frequency $i \geq 2$. When we remove S, the frequency of u decreases by one. Thus, the size of the subproblem

decreases by Δv_i. The overall reduction due to the reduction of the frequencies is at least

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

Finally, the total reduction of the size of S_{out} is

$$w_{|S|} + \sum_{i=2}^{6} m_i \Delta v_i.$$

Now consider the subproblem S_{in}. The size of S_{in} decreases by $w_{|S|}$ because of the removal of S. Since we also remove all elements from S, we also get the reduction of size

$$\sum_{i\geq 2} m_i v_i = \sum_{i=2}^{6} m_i v_i + m_{\geq 7}.$$

Here $m_{\geq 7}$ is the number of elements with frequency at least 7. Let S' be the set sharing the element u with S ($S' \cap S \neq \emptyset$). Note that $|S'| \leq |S|$. When we remove u, the cardinality of S' is reduced by one. This implies a reduction of size S_{in} by $\Delta w_{|S'|} \geq \Delta w_{|S|}$. Thus the overall reduction of the size of S_{in} due to the reduction of the cardinalities of the sets S' is at least

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

Finally, the total reduction of the size of S_{in} is

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

Putting all together, for all possible values of $|S| \geq 3$ and for all values m_i such that

$$\sum_{i=2}^{6} m_i + m_{\geq 7} = |S|$$

(except if $|S| = 3$, then we choose only those sets of m_i where $m_{\geq 7} \neq 0$), we have the following set of recursions

$$P_h(k) \leq P_h(k - \Delta k_{\text{out}}) + P_h(k - \Delta k_{\text{in}}),$$

where

$$\Delta k_{\text{out}} = w_{|S|} + \sum_{i=2}^{6} m_i \Delta v_i,$$

$$\Delta k_{\text{in}} = w_{|S|} + \sum_{i=2}^{6} m_i v_i + m_{\geq 7} + \Delta w_{|S|} \left(\sum_{i=2}^{6} (i-1) m_i + 6 \cdot m_{\geq 7} \right).$$

Since $\Delta w_{|S|} = 0$ for $|S| \geq 7$, we have that each recurrence with $|S| \geq 8$ is "dominated" by some recurrence with $|S| = 7$. Thus we restrict our attention only to the cases where $3 \leq |S| \leq 7$. We need to consider a large number of recursions (1653). For every fixed 9-tuple $(w_1, w_2, w_3, w_4, w_5, v_2, v_3, v_4, v_5)$ the number $P_h(k)$ is upper bounded by α^{k-h}, where α is the largest number from the set of real roots of the set of equations

$$\alpha^k = \alpha^{k-\Delta k_{\text{out}}} + \alpha^{k-\Delta k_{\text{in}}}$$

corresponding to the different combinations of values $|S|$ and m_i. Thus to estimate $P_h(k)$ we need to choose the weights w_i and v_j minimizing α.

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

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

After performing branching the algorithm calls the `countPW` algorithm. Thus the total running time of the algorithm on an instance of measure k is

$$\mathcal{O}^* \left(\sum_{h \in K} \alpha^{k-h} \cdot \beta^h \right) = \mathcal{O}^*(\max\{\alpha, \beta\}^k).$$

Here $\mathcal{O}(\beta^h n^{O(1)})$ is the running time of the `countPW` algorithm on a problem of size h. So we need to choose the weights w_i and v_j minimizing both α and β. To estimate the running time of `countPW` we use the idea of measure and conquer combined with linear programming.

Let us remind that `countPW` is called on an instance of #MWSC with all sets of size at most 3 and elements of frequency at most 6. There are no elements of frequency 1. Let $\mathcal{S}$ be an instance of set cover of measure k and let $G_{\mathcal{S}}$ be its incidence graph. Then $G_{\mathcal{S}}$ is a bipartite graph with the bipartition $(\mathcal{X} = \mathcal{S}, \mathcal{Y} = \mathcal{U}(\mathcal{S}))$. By Lemma 6, the running time of the dynamic programming algorithm on $G_{\mathcal{S}}$ is $\mathcal{O}(\beta^k n^{O(1)}) = \mathcal{O}(2^{\mathbf{pw}(G_{\mathcal{S}})} n^{O(1)})$. Let us remind that both constants α and β depend on the choice of the weights in the measure function. In the remaining part of the proof we show how to choose the weights that balance the branching and dynamic programming parts of the algorithm.

We denote by $\mathcal{X}_i$ all vertices of $\mathcal{X}$ of degree i. Let $x_i = |\mathcal{X}_i|$. We define $\mathcal{Y}_j \subseteq \mathcal{Y}$ and y_j in the same way for every $j \in \{2, \ldots, 6\}$. We need the following lemma.

We need to evaluate the running time of `countPW` on an instance of $\mathcal{S}$ of measure k. This gives us the following:

$$k = k(\mathcal{S}) = \sum_{i=1}^{3} w_i x_i + \sum_{j=2}^{5} v_j y_j + y_6. \tag{4}$$

Here the values w_i and v_j are taken from the analysis of the `countMWSC` algorithm. By counting edges of $G_{\mathcal{S}}$, we arrive at the second condition

$$x_1 + 2x_2 + 3x_3 = \sum_{j=2}^{6} j \cdot y_j. \tag{5}$$

By Lemma 1,

$$\mathbf{pw}(G_{\mathcal{S}}) \leq \frac{1}{6}(x_3 + y_3) + \frac{1}{3}y_4 + \frac{13}{30}y_5 + \frac{23}{45}y_6. \tag{6}$$

Combining (4), (5), and (6) we conclude that the pathwidth of $G_{\mathcal{S}}$ is at most the maximum of the following linear function

$$\frac{1}{6}(x_3 + y_3) + \frac{1}{3}y_4 + \frac{13}{30}y_5 + \frac{23}{45}y_6 \to \max$$

subject to the following constraints:

$$\begin{aligned}
&\text{measure: } \sum_{i=1}^{3} w_i x_i + \sum_{j=2}^{5} v_j y_j + y_6 = k,\\
&\text{edges: } x_1 + 2x_2 + 3x_3 = \sum_{j=2}^{6} j \cdot y_j,\\
&\text{variables: } x_i \geq 0,\ i \in \{1, 2, 3\},\\
&\qquad\qquad y_j \geq 0,\ j \in \{2, \ldots, 6\}.
\end{aligned}$$

The running time of `countPW` is $\mathcal{O}^*(2^{\mathbf{pw}(G)})$. Thus everything boils down to finding the measure that minimizes the maximum of α and the objective function of the LP obtained from the pathwidth bounds. Finding such weights is an interesting (and nontrivial) computational problem on its own; see [8].

We numerically obtain the following values of the weights.

$$w_i = \begin{cases} 0.1039797, & \text{if } i = 1,\\ 0.4664084, & \text{if } i = 2,\\ 0.8288271, & \text{if } i = 3,\\ 0.9429144, & \text{if } i = 4,\\ 0.9899772, & \text{if } i = 5, \end{cases} \quad \text{and} \quad v_i = \begin{cases} 0.5750176, & \text{if } i = 2,\\ 0.7951411, & \text{if } i = 3,\\ 0.9165365, & \text{if } i = 4,\\ 0.9771879, & \text{if } i = 5. \end{cases}$$

With these weights the optimum of LP is obtained on $x_3 = 0.7525$ and $y_6 = 0.3762$ and all other variables equal to 0.

This gives us the total running time $\mathcal{O}(1.2464^{k(\mathcal{S})}) = \mathcal{O}(1.2464^{|\mathcal{U}|+|\mathcal{S}|})$. The space used by the algorithm during the dynamic programming part is bounded by $\mathcal{O}(1.2464^{|\mathcal{U}|+|\mathcal{S}|})$.

The four worst case recurrences for the branching algorithm are listed in Table 1.

This finalizes the proof. □

Table 1 Worst case recurrences for the branching algorithm

$\|S\|$	m_2	m_3	m_4	m_5	m_6	$m_{\geq 7}$
4	0	0	0	4	0	0
4	0	0	0	0	4	0
5	0	0	0	0	0	5
6	0	0	0	0	0	6

As we mentioned already, #MWDS can be reduced to #MWSC by imposing $\mathcal{U} = V$ and $\mathcal{S} = \{N[v] \mid v \in V\}$. The size of the corresponding #MWSC instance is at most $2n$, where n is the number of vertices in G. Thus, we have

Corollary 2 *The* #MWDS *problem can be solved in* $\mathcal{O}(1.2464^{2n}) = \mathcal{O}(1.5535^n)$ *time.*

6 Application to Parameterized Algorithms

In this section we show the versatility of our techniques by applying them to parameterized problems.

6.1 Weighted Vertex Cover

Here we apply our technique to design a simple fixed parameter tractable algorithm for the parameterized version of WEIGHTED VERTEX COVER problem.

- k-WEIGHTED VERTEX COVER (k-WVC): Given a graph $G = (V, E)$, a weight function $w : V \to \mathbb{R}^+$ such that for every vertex v, $w(v) \geq 1$ and $k \in \mathbb{R}^+$, find a vertex cover of weight at most k, where the weight of a vertex cover C is $w(C) = \sum_{v \in C} w(v)$.

Now, we present an algorithm that combines Branch & Reduce and dynamic programming on graphs of bounded treewidth. It is well known that a minimum vertex cover can be found in time $\mathcal{O}(2^\ell n^{O(1)})$ in a graph of treewidth at most ℓ.

Proposition 5 [2] *Given a graph G with weights on its vertices and a tree decomposition of G of width at most ℓ, a minimum weighted vertex cover of G can be found in time $\mathcal{O}(2^\ell n^{O(1)})$.*

We need *kernelization* for our algorithm for weighted vertex cover. The main idea of *kernelization* is to replace a given instance (I, k) by a simpler instance (I', k') using some *data reduction rules* in polynomial time such that (I, k) is a yes-instance if and only if (I', k') is a yes-instance and $|I'|$ is bounded by a function of k alone. We state the kernelization proposition of [25] that we use in our algorithm.

Proposition 6 [25] *Let $G = (V, E)$ be a graph, $w : V \to \mathbb{R}^+$ such that for every vertex v, $w(v) \geq 1$ and $k \in \mathbb{R}^+$. There is an algorithm that in time $\mathcal{O}(kn + k^3)$ either concludes that G has no vertex cover of weight at most k, or outputs a kernel of size at most $2k$.*

First we apply Proposition 6 to obtain a kernel of size at most $2k$. Then, as long as the maximum degree of the graph is at least 4, the algorithm branches on a vertex v of maximum degree; two subproblems are created according to the following rules:

(1) add v to the partially constructed vertex cover and delete v from the graph;
(2) add $N(v)$ to the partially constructed vertex cover and delete $N[v]$ from the graph.

If the maximum degree of the graph is at most 3, then by Proposition 1, a tree decomposition of small tree width (tw) can be found on the kernel of size $2k$ in polynomial time and we can use a $\mathcal{O}(2^{tw}n^{O(1)})$ dynamic programming algorithm to solve k-WEIGHTED VERTEX COVER. The correctness is clear from the presentation and the running time of the algorithm is dominated by the following recurrences on $T(k)$.

$$T(k) \leq T(k-1) + T(k-4) \quad \text{[Branching Step]},$$
$$T(k) \leq 2^{2k/6}n^{O(1)} \quad \text{[Treewidth Step]}.$$

Though the gap between the solutions of the above two recurrences are huge, it is hard to balance them. The problem is that the known bound on the size of the kernel remains fixed even though the average degree or the maximum degree of the graph decreases. This results in the following theorem.

Theorem 5 *k-WVC on a graph on n vertices can be solved in time $\mathcal{O}(1.3803^k n^{O(1)})$ and space $\mathcal{O}(1.2599^k n^{O(1)})$.*

This simple algorithm is comparable to the best known parameterized algorithm for weighted vertex cover which runs in time $\mathcal{O}(1.3788^k n^{O(1)})$ and space $\mathcal{O}(1.3603^k n^{O(1)})$ [25].

6.2 Parameterized Edge Dominating Set and Its Variants

In this subsection we show that Branching & Global Application of Width Parameters can be used to obtain the fastest known parameterized algorithm for the following problem.

- k-WEIGHTED EDGE DOMINATING SET (k-WEDS): Given a graph $G = (V, E)$, a weight function $w : E \to \mathbb{R}^+$ such that for every edge e, $w(e) \geq 1$ and $k \in \mathbb{R}^+$, find a set of edges $D \subseteq E$ of weight $w(D) = \sum_{e \in D} w(e)$ at most k such that every edge of $E \setminus D$ is adjacent to an edge in D.

Observe that if a graph G has an edge dominating set of weight at most k then it has a vertex cover of weight at most $p = 2k$. As in the algorithm for MMM, we construct a partial vertex cover C of G by branching on vertices of maximum degree. As observed by Fernau [9], if $G \setminus C$ has maximum degree one, a corresponding edge dominating set for G can be found in polynomial time. Using this and branching on vertices of degree at least 2, Fernau obtains an algorithm with running time $\mathcal{O}(2.6181^k n^{O(1)})$.

Here we branch on vertices of degree at least 3. That is we pick a vertex v of degree at least 3 and include either v or $N(v)$ in C. This gives us the following recurrence on p: $T(p) = T(p-1) + T(p-3)$, which solves to $\mathcal{O}(1.465572^p) = \mathcal{O}(2.1480^k)$.

Now, suppose that the algorithm has reached a branch node (G, H, C) of the recurrence tree and $\Delta(H) \leq 2$. Then by Lemma 4, the pathwidth of the graph G is bounded by $|C| + 1$. Again, we use a different strategy based on the size of $|C|$ at the branch node. If $|C| \leq \alpha p$ (α to be determined later) then we compute a path decomposition of width ℓ and apply an algorithm with running time $\mathcal{O}(3^\ell n^{O(1)})$ similar to the algorithm of Theorem 3 to obtain a minimum edge dominating set. Otherwise, the algorithm continues branching on vertices of degree 2 of H in time $1.6181^{p-\alpha p}$. To obtain the optimal value of α, we solve the equation $1.465572^{\alpha p} 1.6181^{p-\alpha p} = 3^{\alpha p}$ and obtain $\alpha = 0.4018$. This gives us a running time of $\mathcal{O}(1.55501^p n^{O(1)}) = \mathcal{O}(2.4181^k n^{O(1)})$ for k-Edge Dominating Set.

We observed in Sect. 4.2 that an exact algorithm for Edge Dominating Set also implies exact algorithms for Minimum Maximal Matching and Matrix Domination. Thus, we obtain the following result for k-Weighted Edge Dominating Set and its related problems.

Theorem 6 *k-Weighted Edge Dominating Set, k-Minimum Maximal Matching and k-Matrix Domination can be solved in time $\mathcal{O}(2.4181^k n^{O(1)})$.*

7 Conclusion

Branching and dynamic programming on graphs of bounded treewidth are very powerful techniques to design efficient exact algorithms. In this paper, we combined these two techniques in different ways and obtained improved exact algorithms for #MWDS, MMM and its variants. We also applied the technique to design fixed parameter tractable algorithms and obtained fast algorithms for k-WVC and k-WEDS which also shows the versatility of our technique. The most important aspects of this technique are that the resulting algorithms are very elegant and simple while at the same time the analysis of these algorithms is non-trivial.

It would be interesting to find some other applications of the techniques presented here in the design of exact exponential time algorithms and fixed parameter tractable algorithms.

References

1. Angelsmark, O., Jonsson, P.: Improved algorithms for counting solutions in constraint satisfaction problems. In: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP 2003), pp. 81–95 (2003)
2. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybern. **11**, 1–21 (1993)
3. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Sci. **209**, 1–45 (1998)
4. Byskov, J.M.: Enumerating maximal independent sets with applications to graph colouring. Oper. Res. Lett. **32**, 547–556 (2004)

5. Chandran, L.S., Grandoni, F.: Refined memorization for vertex cover. Inf. Process. Lett. **93**, 125–131 (2005)
6. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. J. Algorithms **41**, 280–301 (2001)
7. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
8. Eppstein, D.: Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. ACM Trans. Algorithms **2**, 492–509 (2006)
9. Fernau, H.: Edge dominating set: efficient enumeration-based exact algorithms. In: Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC 2006). Lecture Notes in Computer Science, vol. 4169, pp. 142–153. Springer, Berlin (2006)
10. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Inf. Process. Lett. **97**, 191–196 (2006)
11. Fomin, F.V., Grandoni, F., Kratsch, D.: Some new techniques in design and analysis of exact (exponential) algorithms. Bull. EATCS **87**, 47–77 (2005)
12. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: domination—a case study. In: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005). Lecture Notes in Computer Science, vol. 3580, pp. 191–203. Springer, Berlin (2005)
13. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Bounding the number of minimal dominating sets: a measure and conquer approach. In: Proceedings of the 16th Annual International Symposium on Algorithms and Computation (ISAAC 2005). Lecture Notes in Computer Science, vol. 3827, pp. 573–582. Springer, Berlin (2005)
14. Fomin, F.V., Kratsch, D., Woeginger, G.J.: Exact (exponential) algorithms for the dominating set problem. In: Proceedings of the 30th Workshop on Graph Theoretic Concepts in Computer Science (WG 2004). Lecture Notes in Computer Science, vol. 3353, pp. 245–256. Springer, Berlin (2005)
15. Fomin, F.V., Gaspers, S., Saurabh, S.: Branching and treewidth based exact algorithms. In: Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC 2006). Lecture Notes in Computer Science, vol. 4288, pp. 16–25. Springer, Berlin (2006)
16. Fürer, M., Kasiviswanathan, S.P.: Algorithms for counting 2-SAT solutions and colorings with applications. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 33 (2005)
17. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman (1979)
18. Gavril, F., Yannakakis, M.: Edge dominating sets in graphs. SIAM J. Appl. Math. **38**, 364–372 (1980)
19. Grandoni, F.: A note on the complexity of minimum dominating set. J. Discrete Algorithms **4**, 209–214 (2006)
20. Iwama, K.: Worst-case upper bounds for k-SAT. Bull. EATCS **82**, 61–71 (2004)
21. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Inf. Process. Lett. **27**, 119–123 (1988)
22. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Algorithms based in treewidth of sparse graphs. In: Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005). Lecture Notes in Computer Science, vol. 3787, pp. 385–396. Springer, Berlin (2005)
23. Moon, J.W., Moser, L.: On cliques in graphs. Israel J. Math. **3**, 23–28 (1965)
24. Niedermeier, R., Rossmanith, P.: Upper bounds for vertex cover further improved. In: Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science (STACS 1999). Lecture Notes in Computer Science, vol. 1563, pp. 561–570. Springer, Berlin (1999)
25. Niedermeier, R., Rossmanith, P.: On efficient fixed-parameter algorithms for weighted vertex cover. J. Algorithms **47**, 63–77 (2003)
26. Raman, V., Saurabh, S., Sikdar, S.: Efficient exact algorithms through enumerating maximal independent sets and other techniques. Theory Comput. Syst. (to appear)
27. Randerath, B., Schiermeyer, I.: Exact algorithms for MINIMUM DOMINATING SET. Technical Report zaik-469. Zentrum für Angewandte Informatik Köln, Germany (2004)
28. Schöning, U.: Algorithmics in exponential time. In: Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS 2005). Lecture Notes in Computer Science, vol. 3404, pp. 36–43. Springer, Berlin (2005)
29. Scott, A.D., Sorkin, G.B.: Linear-programming design and analysis of fast algorithms for Max 2-Sat and Max 2-CSP, arXiv.org, cs/0604080, 2006

30. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theor. Comput. Sci. **348**, 357–365 (2005)
31. Woeginger, G.: Exact algorithms for NP-hard problems: a survey. In: Combinatorial Optimization—Eureka, You Shrink! Lecture Notes in Computer Science, vol. 2570, pp. 185–207. Springer, Berlin (2003)
32. Woeginger, G.: Space and time complexity of exact algorithms: some open problems. In: Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC 2004). Lecture Notes in Computer Science, vol. 3162, pp. 281–290. Springer, Berlin (2004)