REPORTS
IN
INFORMATICS

ISSN 0333-3590

Counting minimum weighted dominating sets

Fedor V. Fomin, Alexey A. Stepanov

REPORT NO 332 October 2006

Counting minimum weighted dominating sets

Fedor V. Fomin
Department of Informatics,
University of Bergen,
PO Box 7803,
5020 Bergen, Norway
fomin@ii.uib.no

Alexey A. Stepanov
Department of Informatics,
University of Bergen,
PO Box 7803,
5020 Bergen, Norway
ljosha@ljosha.org

Abstract

We show how to count all minimum weighted dominating sets of a graph on n vertices in time $\mathcal{O}(1.5535^n)$.

1 Introduction

The story of exact (exponential-time) algorithms for hard problems dates back to the sixties and seventies but especially the last decade has seen a growing interest in new techniques for such type of algorithms. We refer to the following recent surveys [8, 21, 23] for an overview of the field.

Counting problem is a natural extension of a decision problem, where instead of deciding on the existence of a solution, the task is to find the number of solutions. Valiant [22] defined the class #P and showed that computing the permanent is #P-complete. Many NP-complete as well as some problems in P can have their counting versions to be #P-hard. In particular, counting minimum dominating set is #P-hard [15] even when restricted to planar instances. There is a lot of research going on counting problems and the complexity class #P. (See the book by Jerrum [16] for an introduction.)

While exact algorithms for many NP-complete decision problems were studied quite intensively, there are not so many results on exact algorithms for #P-complete problems in the literature. For a long time the only known exact algorithm for an #P-complete problem was time $2^n \cdot n^{\mathcal{O}(1)}$ counting perfect matching in bipartite graph algorithm due to Ryser [20]. There are known exact algorithms for different counting versions of satisfiability problem like counting maximum weighted models for 2SAT and 3SAT [5, 12, 24], CSP [1], and counting maximum weighted independent sets [4, 6]. Very recently time $2^n \cdot n^{\mathcal{O}(1)}$ algorithm computing chromatic polynomial (counting all k-colorings for each k) was obtained in [2, 17].

Previous results. The dominating set problem is one of the classical NP-complete graph optimization problem which fits into the broader class of domination and covering problems. Hundreds of papers have been written on them (see e.g. the survey [14] by Haynes et al.). Recently, several groups of authors independently obtained exact algorithms that solve minimum dominating set problem in a graph on n vertices faster than the trivial $2^n \cdot n^{O(1)}$-time brute force algorithm [11, 13, 19]. The fastest known algorithm computes a minimum dominating set of a graph in time $\mathcal{O}(1.5137^n)$ [7]. The algorithm from [7] cannot be used to compute a dominating set of minimum weight (in a weighted graph), however, as it was observed in [9], 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.

Our results. In this paper 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 of the dominating set problem to the instance of a set cover problem (this is the idea used by Grandoni [13] for decision problem) and perform branching on large sets and sets of size three containing elements of high degree. When branching is complete, we turn the instance of set cover into an instance of red/blue domination on bipartite graphs and use dynamic programming to count all solutions.

The novel and the most difficult part of the paper is the analysis of the algorithm. To analyze the running time we need to 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 "balance" branching and dynamic programming parts of the algorithm. To choose the right measure we express the bounds on pathwidth as a linear program.

Combining branching with dynamic programming is a general approach that works for many decision and counting problems [6, 18] and our technique can be used to improve the analysis of many algorithms of this type.

Finally, let us remark that the running time $\mathcal{O}(1.5535^n)$ of our algorithm counting weight dominating sets is even (slightly) better than the best known before this paper the running time of the minimum weighted dominating set algorithm from [9].

2 Preliminaries

Let $G = (V, E)$ be an n-vertex undirected, simple graph without loops. We denote by $\Delta(G)$ the maximum vertex degree in G. For a vertex $v \in V$ we denote the set of its neighbors by $N(v)$ and its *closed neighborhood* by $N[v] = N(v) \cup \{v\}$.

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. Given a weight function $w : V \longrightarrow \mathbb{R}^+$ the *Minimum Weighted Dominating Set* problem (`MWDS`) asks to 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.

Let $\mathcal{U}$ be a set of elements and $\mathcal{S}$ be a collection of non-empty subsets of $\mathcal{U}$. Given a weight function $w : \mathcal{S} \longrightarrow \mathbb{R}^+$ the *Minimum Weighted Set Cover* problem (`MWSC`) asks to 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 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 $\mathrm{freq}(u)$.

#`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 weight for $S \in \mathcal{S}$ 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 weighted dominating set problem called *minimum red/blue weighted dominating set* (`RBWDS`) problem. For a bipartite graph $G = (V, E)$ with a bipartition $V = V_{Red} \cup V_{Blue}$ and a weight function $w : V \longrightarrow \mathbb{R}^+$, a subset $D \subseteq V_{Red}$ is *red-blue dominating set* if every vertex in V_{Blue} is adjacent to a vertex of D. `RBWDS` problem is to determine the minimum weight of a red/blue dominating set in G.

With an instance $(\mathcal{U}, \mathcal{S}, w)$ of `MWSC` one can associate an *incidence graph* $G_{\mathcal{S}}$, which is a bipartite graph on a vertex set $\mathcal{S} \cup \mathcal{U}$ with a 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 $V_{Red} = \mathcal{S}$ and $V_{Blue} = \mathcal{U}$) dominating set of weight k.

Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\langle \{X_i \mid i \in I\}, T\rangle$, where each X_i is a subset of V, called a *bag*, and T is a tree with the elements of I as vertices. The following three properties must hold:

1. $\bigcup_{i \in I} X_i = V$.
2. For every edge $(u, v) \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$.
3. For all $i, j, k \in I$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of $\langle \{X_i \mid i \in I\}, T\rangle$ equals $\max\{|X_i| \mid i \in I\} - 1$. The *treewidth* of G is the minimum k such that G has a tree decomposition of width k. A tree decomposition is called a *path decomposition* if T is a path. Accordingly, the *pathwidth* of G is the minimum k such that G has a path decomposition of width k. We denote by $\mathbf{pw}(G)$ the pathwidth of G.

We need the following result which can be obtained by a standard dynamic programming techniques. (The proof of this lemma is moved to Appendix.)

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

3 Algorithm for counting minimum weighted set covers

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

Lemma 2. *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.*

```
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.
if $|\mathcal{S}| = 0$ then
    return (0,1);
if $\exists u \in \mathcal{U}(\mathcal{S}) : \text{freq}(u) = 1$, Let $u \in S'$ then
    return countMWSC(remove($S'$, $\mathcal{S}$), $w$);
Pick $S \in \mathcal{S}$ 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
    return countPW($\mathcal{S}, w$);
else
    $(w_{in}, n_{in})$ = countMWSC(remove($S, \mathcal{S}$), $w$);
    $(w_{out}, n_{out})$ = countMWSC($\mathcal{S} \setminus \{S\}$, $w$);
    $w_{in} = w_{in} + w(S)$;
    if $w_{in} < w_{out}$ then
        return $(w_{in}, n_{in})$;
    else if $w_{in} = w_{out}$ then
        return $(w_{in}, n_{in} + n_{out})$;
    else
        return $(w_{out}, n_{out})$;
```

Figure 1: countMWSC$(\mathcal{S}, w)$

Let us go through the algorithm `countMWSC`; see Figure 1. First, if $|\mathcal{S}| = 0$ then the size of `MWSC` is 0 and the number of such set covers is 1. Otherwise (lines 3–4), the algorithm tries to reduce the size of the problem by checking whether condition of Lemma 2 is applicable. Specifically, if there is an element $u \in S'$ of frequency one, then we should put S' into 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 2 does not apply, then we take 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 subproblems. First subproblem (`remove(`S`,`$\mathcal{S}$`)`,w) corresponds to the case where S belongs to the minimum set cover. Whereas in $(\mathcal{S} \setminus \{S\}, w)$ subproblem S does not belong to the minimum set cover.

And finally we compare the weights of two subproblems and return total weight and number (lines 12–17).

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.

- Constructs the incidence graph $G_{\mathcal{S}}$ of $\mathcal{S}$;
- Counts 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 perform dynamic programming algorithm described in Lemma 1);
- Counts the number of minimum set covers of $\mathcal{S}$ from the number of red/blue dominating sets of $G_{\mathcal{S}}$.

4 Analysis of `countMWSC` algorithm

In this section we show that the running time of the algorithm `countMWSC` is $\mathcal{O}^*(1.2464^{|\mathcal{S}|+|\mathcal{U}|})$. The analysis is based on *Measure & Conquer* technique [8, 7]. The analysis of the branching part of the algorithm is quite similar to analysis from [7].

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

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

Intuitively, $\Delta w_i (\Delta v_i)$ is a 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 introduced is removed before next branching. And thus we get the total reduction $1 - (1 - v_2) = v_2$.

Theorem 1. *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 one of 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 5. If $|S| \leq 3$ and for every $S \in \mathcal{S}$ the degree of all its elements is at most 6 (line 6), no subproblem is generated. Otherwise, we branch on two subproblems $S_{out} = (w_{out}, n_{out})$ and $S_{in} = (w_{in}, n_{in})$.

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 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 the 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

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

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

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

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

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

where

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

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

Since $\Delta w_{|S|} = 0$ for $|S| \geq 7$, we have that each recursion with $|S| \geq 8$ is "dominated" by some recurrence with $|S| = 7$. Thus we restrict our attention only to the cases $3 \leq |S| \leq 7$. 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

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

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 subproblems solved. 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 `countPW` algorithm. Thus the total running time of the algorithm on an instance of measure k is

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

Here $\mathcal{O}(\beta^h)$ is the running time of `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` algorithm we use the idea of measure and conquer applied to linear programming.

Let us remind that `countPW` is called on an instance of `MWSC` problem 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 1, the running time of dynamic programming algorithm on $G_{\mathcal{S}}$ is $\mathcal{O}(\beta^k) = \mathcal{O}(2^{\mathbf{pw}(G_{\mathcal{S}})})$. 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 branching and dynamic programming parts of the algorithm.

We denote by $\mathcal{X}_i$ all vertices from $\mathcal{X}$ 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.

Lemma 3. *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 6,*

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

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

Proof. Let $G = (V, E)$ be a graph with 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 at most by a logarithmic factor. So we assume that G has no vertices of degree 1 and 2 (otherwise we contract the corresponding edges).

It is known [6, 10] that for every $\varepsilon > 0$, there exists an integer n_ε such that for every graph of maximum degree 4 with $n > n_\varepsilon$ vertices,

$$\mathbf{pw}(G) \le \frac{n_3}{6} + \frac{n_4}{3} + \varepsilon n.$$

We prove Lemma by induction on the number of vertices of degree at least 5. We start with n_5. If $n_5 = n_6 = 0$ then $\Delta(G) \le 4$ and we apply result mentioned above. Let us assume that for some $n_5 \ge 1$ the statement of Lemma holds for all graphs with at most $n_5 - 1$ vertices of degree 5, no vertices of degree 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 assume first that the graph $G - v$ is not 5-regular. It is clear that $\mathbf{pw}(G) \le \mathbf{pw}(G - v) + 1$. For $j \in \{3, \dots, 5\}$ we denote by m_j the number of degree j neighbors of v. By induction assumption,

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

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

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

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

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

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 $\mathbf{pw}(G) \le \mathbf{pw}(G - u - v) + 2$, we have that

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

Thus we prove that Lemma holds for all non 5-regular graphs. Since removal of one vertex changes the pathwidth by additive factor at most 1, for sufficiently large n this additive factor is dominated by εn, and we conclude that 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 v of degree 6 has 5 neighbors of degree 6 and one neighbor of degree 5. $\square$

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

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

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

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

$\lvert S\rvert$	$(m_2, m_3, m_4, m_5, m_6, m_{\geq 7})$
4	$(0,0,0,4,0,0)$
4	$(0,0,0,0,4,0)$
5	$(0,0,0,0,0,5)$
6	$(0,0,0,0,0,6)$

Table 1: Worst case recurrences for the branching algorithm

By Lemma 3,

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

Combining (1),(2), and (3) 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:

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

The running time of `countPW` is $\mathcal{O}^*(2^{\mathbf{pw}(G)})$. Thus everything boils up to finding the measure that minimize the maximum of α and maximum of LP obtained from pathwidth bounds. Finding of such weights is an interesting (and nontrivial) computational problem on its own. To find the weights we use a modification of random search with plugged LP solver.

We numerically obtained 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 such 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 exponential space is used by the algorithm during the dynamic programming part and thus is bounded by $\mathcal{O}(1.2464^{|\mathcal{U}|+|\mathcal{S}|})$. Four worst case recurrencies for the branching algorithm are listed in Table 1.

This finalizes the proof. □

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 #`MWSC` instance obtained is at most $2n$, where n is the number of vertices in G. Thus, we have

Corollary 1. *The #`MWDS` problem can be solved in $\mathcal{O}(1.2464^{2n}) = \mathcal{O}(1.5535^n)$ time.*

5 Conclusions and open problems

In this paper we have used dynamic programming on bounded treewidth techniques to speed up branching algorithm counting weighted dominating sets. The running time of our algorithm can be slightly improved by considering more detailed bounds on pathwidth of bipartite graphs with different coefficients for vertices adjacent to degree two and three vertices. However in this case the arguments become much more technical and we do not include them in this extended abstract.

Another general technique to speed up branching algorithms (by using exponential space) is *memorization*, see [8]. Both techniques have many similarities and it would be interesting to understand in which cases each of the techniques is more efficient.

The best known algorithm for the decision version of minimum dominating set is not significantly faster than our counting algorithms. Similar strange effect was observed by Magnus Wahlström (private communication) for different SAT problems. Thus despite the fact that #P complete problems seems to be much more difficult than NP complete problems, the running time of best known algorithms for many decision and counting problems do not differ too much. Is there any reasonable explanation to this phenomena? Is this because faster exponential algorithms for decision problems are still to be found or this is because in the world of exponential time algorithms the gaps between different complexity classes are small?

Acknowledgement. We thank Serge Gaspers for helpful comments.

References

[1] O. ANGELSMARK AND P. JONSSON, *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), vol. 2833 of LNCS, Springer, 2003, pp. 81–95. 1

[2] A. BJÖRKLUND AND T. HUSFELDT, *Inclusion-exclusion algorithms for counting set partitions*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), IEEE, 2006, p. to appear. 1

[3] H. L. BODLAENDER, *A partial k-arboretum of graphs with bounded treewidth*, Theoretical Computer Science, 209 (1998), pp. 1–45. 4

[4] V. DAHLLÖF AND P. JONSSON, *An algorithm for counting maximum weighted independent sets and its applications*, in 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), ACM and SIAM, 2002, pp. 292–298. 1

[5] V. DAHLLÖF, P. JONSSON, AND M. WAHLSTRÖM, *Counting models for 2SAT and 3SAT formulae*, Theoret. Comput. Sci., 332 (2005), pp. 265–291. 1

[6] F. V. FOMIN, S. GASPERS, AND S. SAURABH, *Branching and treewidth based exact algorithms*, in Proceedings of the 17th Annual International Symposium on Algorithms and Computation (ISAAC 2006), Lecture Notes in Comput. Sci., Springer, Berlin, 2006, p. to appear. 1, 4

[7] 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 Lecture Notes in Comput. Sci., Springer, Berlin, 2005, pp. 191–203. 1, 4

[8] ——, *Some new techniques in design and analysis of exact (exponential) algorithms*, Bulletin of the EATCS, 87 (2005), pp. 47–77. 1, 4, 5

[9] F. V. FOMIN, F. GRANDONI, A. V. PYATKIN, AND A. A. STEPANOV, *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), vol. 3827 of Lecture Notes in Comput. Sci., Springer, Berlin, 2005, pp. 573–582. 1

[10] F. V. FOMIN AND K. HØIE, *Pathwidth of cubic graphs and exact algorithms*, Information Processing Letters, 97 (2006), pp. 191–196. 4

[11] 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 Lecture Notes in Comput. Sci., Springer, Berlin, 2004, pp. 245–256. 1

[12] M. FÜRER AND S. P. KASIVISWANATHAN, *Algorithms for counting 2-SAT solutions and colorings with applications*, in Electronic Colloquium on Computational Complexity (ECCC), vol. 33, 2005. 1

[13] F. GRANDONI, *A note on the complexity of minimum dominating set*, Journal of Discrete Algorithms, 4 (2006), pp. 209–214. 1

[14] T. W. HAYNES AND S. T. HEDETNIEMI, eds., *Domination in graphs*, Marcel Dekker Inc., New York, 1998. 1

[15] H. B. HUNT, III, M. V. MARATHE, V. RADHAKRISHNAN, AND R. E. STEARNS, *The complexity of planar counting problems*, SIAM J. Comput., 27 (1998), pp. 1142–1167. 1

[16] M. JERRUM, *Counting, sampling and integrating: algorithms and complexity*, Lectures in Mathematics ETH Zürich, Birkhäuser Verlag, Basel, 2003. 1

[17] M. KOIVISTO, *An $O(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), IEEE, 2006, p. to appear. 1

[18] D. MÖLLE, S. RICHTER, AND P. ROSSMANITH, *Enumerate and expand: Improved algorithms for connected vertex cover and tree cover*, in Proceedings of the First International Computer Science Symposium in Russia (CSR 2006), vol. 3967 of LNCS, Springer, 2006, pp. 270–280. 1

[19] B. RANDERATH AND I. SCHIERMEYER, *Exact algorithms for MINIMUM DOMINATING SET*, Technical Report zaik-469, Zentrum für Angewandte Informatik Köln, Germany, 2004. 1

[20] H. J. RYSER, *Combinatorial mathematics*, The Carus Mathematical Monographs, No. 14, Published by The Mathematical Association of America, 1963. 1

[21] U. SCHÖNING, *Algorithmics in exponential time*, in Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS 2005), vol. 3404 of Lecture Notes in Comput. Sci., Springer, Berlin, 2005, pp. 36–43. 1

[22] L. G. VALIANT, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201. 1

[23] G. WOEGINGER, *Exact algorithms for NP-hard problems: A survey*, in Combinatorial Optimization - Eureka, you shrink!, vol. 2570 of Lecture Notes in Comput. Sci., Springer-Verlag, Berlin, 2003, pp. 185–207. 1

[24] W. ZHANG, *Number of models and satisfiability of sets of clauses*, Theoretical Computer Science, 155 (1996), pp. 277–288. 1

A Appendix: Proof of Lemma 1

Proof. Let $G = (V, E)$ be a bipartite graph $G = (V, E)$ with a bipartition $V = V_{Red} \cup V_{Blue} = A \cup B$ and let $(X_1, X_2, \dots, X_l)$ be its path decomposition of width p. First we show how to compute a minimum weighted red/blue dominating set in G. It is well known that a path decomposition can be turned into a path decomposition $(X_1, X_2, \dots, X_k)$ where $|X_1| = 1$ and for every $i \in \{1, 2, \dots, k-1\}$ there is $v \in V$ such that either $X_{i+1} = X_i \cup \{v\}$, or $X_{i+1} = X_i \setminus \{v\}$ and that such a construction can be done in linear time.

For every node $i \in \{1, 2, \dots, k\}$, let $S_i \subseteq V$ be the subset of vertices $\bigcup_{j \le i} X_j$ and G_i be the subgraph of G induced by S_i.

In dynamic programming we use two colors: 0 and 1 to represent the state of the vertices in the graph. The meaning behind this coloring is the following. If a vertex from A is colored by 1, then this vertex should be in the dominating set. If it is colored by 0, then it is not in the dominating set. If a vertex from B is colored by 1, then it is dominated by some vertex from S_i. If its color is 0, then this vertex should be dominated by some vertex outside S_i.

More precisely, for every $i \le k$ and every coloring c of X_i, we compute the minimum weight of a red/blue dominating set in G_i where minimum is taking over all red/blue dominating sets in G_i such that

- $(0A)$: The vertices of $A \cap X_i$ colored by 0 are not in the dominating set;
- $(1A)$: The vertices of $A \cap X_i$ colored by 1 are in the dominating set;
- $(0B)$: The vertices of $B \cap X_i$ colored by 0 are not dominated;
- $(1B)$: The vertices of $B \cap X_i$ colored by 1 are dominated.

With every node i of the path decomposition we associate a table. The columns of the table store all possible colorings of the vertices in X_i ($2^{|X_i|}$ columns); and for every coloring $c = (c(x_1), \dots, c(x_{|X_i|})) \in \{0, 1\}^{|X_i|}$, we also keep value $f_i(c)$ which is the minimum weight of a red/blue dominating set for

$$G_i \setminus \{x \in X_i \cap B \mid c(x) = 0\}$$

consistent with this coloring.

The dynamic programming starts from X_1 and continues by calculating the table of node $i + 1$ from the table of node i. The weight of a minimum red/blue dominating set for G is then the minimum number value of f_k of the table associated with the colorings of X_k containing no 0s in $X_k \cap B$.

What follows is a description—for a node i with associated bag X_i—how the values of the associated table are calculated, depending on the type of i.

$\underline{X_1}$ Suppose that $X_1 = \{x\}$. If $x \in A$, then there are two possibilities to color x: 0 and 1. The value of $f_1(c)$ is 0 if $c(x) = 0$ and $f_1(c)$ is 1 otherwise. If $x \in B$, there is only one coloring, $c(x) = 0$, the value of $f_1(c)$ is $w(x)$ in this case.

$\underline{X_{i+1} = X_i \cup \{x\}}$: For each coloring $c' = (c'(x_1), \dots, c'(x_{|X_i|}))$ of X_i we construct a coloring c for X_{i+1} in the following way. There are four cases:

1. $x \in A, c(x) = 0$ and $c(x_j) = c'(x_j)$ for every $j \in [1, |X_i|]$. Thus, $f_{i+1}(c) = f_i(c')$;
2. $x \in A, c(x) = 1$, for every neighbor x_k of $x : x_k \in X_i \cap B$, $c(x_k) = 1$, and $c(x_j) = c'(x_j)$ for other x_j. Thus, $f_{i+1}(c) = f_i(c') + w(x)$;
3. $x \in B$. If there is no neighbor of x in X_i colored 1, then $c(x) = 0$; and $c(x_j) = c'(x_j)$ for every $j \in [1, |X_i|]$. Thus, $f_{i+1}(c) = f_i(c')$;
4. $x \in B$. If there is neighbor of x in X_i colored 1, then $c(x) = 1$; and $c(x_j) = c'(x_j)$ for every $j \in [1, |X_i|]$. Thus, $f_{i+1}(c) = f_i(c')$.

$\underline{X_{i+1} = X_i - \{x\}}$: For every coloring $c' = (c'(x_1), \ldots, c'(x_{|X_{i-1}|}), c'(x))$ of X_i we construct a table for X_{i+1} in the following way. If $c'(x) = 0$ and $x \in B$, then the coloring c' is not valid and $f_{i+1}(c') = +\infty$. Otherwise, $f_{i+1}(c') = f_i(c')$.

Finally, each table can be computed in time $\mathcal{O}(2^p)$ and the overall time complexity of the algorithm is thus $\mathcal{O}(2^p k) = \mathcal{O}(2^p n)$. The last table contains weights of all dominating sets of G. Thus, we conclude that all minimum weighted dominating sets can be counted in $\mathcal{O}(2^p n)$ time. $\square$