Branching and Treewidth Based Exact Algorithms*

Fedor V. Fomin[1], Serge Gaspers[1] and Saket Saurabh[2]

[1] Department of Informatics, University of Bergen,
N-5020 Bergen, Norway.
{fomin|serge}@ii.uib.no
[2] The Institute of Mathematical Sciences,
Chennai 600 113, India.
saket@imsc.res.in

Abstract. Branch & Reduce and dynamic programming on graphs of bounded treewidth are among the most common and powerful techniques used in the design of exact (exponential time) algorithms for NP hard problems. In this paper we discuss the efficiency of *simple* algorithms based on combinations of these techniques. We give several examples of possible combinations of branching and programming which provide the fastest known algorithms for a number of NP hard problems: Minimum Maximal Matching and some variations, counting the number of maximum weighted indepndent sets. We also briefly discuss how similar techniques can be used to design parameterized algorithms. As an example, we give fastest known algorithm solving k-Weighted Vertex Cover problem.

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 [4, 14] 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.

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 [1] 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 $O(c^{\ell}n^{O(1)})$, 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 [4, 5, 9]. For example, a maximum

* Additional support by the Research Council of Norway.

independent set of a graph given with a tree decomposition of width at most ℓ can be found in time $O(2^\ell n)$ (see e.g. [1]). 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. Unfortunately, such a simple approach still provides poor running time mainly because the best known upper bounds on treewidth of graphs with small maximum degree are too large to be useful.

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. 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 the subproblems is crucial). We exemplify our approaches on the following problems.

- Minimum Maximal Matching (MMM): Given a graph G, find a maximal matching of minimum size.
- #Maximum Weighted Independent Set (#MWIS): Given a weighted graph G, count the number of independent sets in G of maximum weight.

For MMM, a number of exact algorithms can be found in the literature. Randerath and Schiermeyer [13] gave an algorithm of time complexity $O(1.44225^m)$ [3] for MMM, where m is the number of edges. Raman et al [12] improved the running time by giving an algorithm of time complexity $O(1.44225^n)$ 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 $O(1.4082^n)$ algorithm for MMM.

There was number of algorithms for #MWIS in the literature [2, 6, 7]. The current fastest algorithm is by Fürer and Kasiviswanathan [6] and runs in $O(1.2461^n)$. All mentioned algorithms are complicated and use many smart tricks (like splitting of a graph into its biconnected components and involved measure) and extensive case analysis.

In this paper we show how a combination of branching and dynamic programming can be used to obtain a simple algorithm solving #MWIS in time $O(1.2431^n)$. This is also the fastest known algorithm to find a maximum weighted independent set in a weighted graph G.

Finally we apply our technique to Parameterized Complexity. Here, we apply our technique to parameterized k-Weighted Vertex Cover.

- 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. The weight of a vertex cover C is $w(C) = \sum_{v \in C} w(v)$.

For k-Weighted Vertex Cover, also known as Real Vertex Cover, Niedermeier and Rossmanith [11] gave two algorithms, one with running time $O(1.3954^k + kn)$ and polynomial space and the other one using time $O(1.3788^k + kn)$ and space $O(1.3630^k)$.

[3] We round the base of the exponent in all our algorithms which allows us to ignore polynomial terms and write $O(c^n n^{O(1)})$ as $O(c^n)$.

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 $O(1.3570^k n)$ for this problem, improving the previous $O(1.3788^k + kn)$ time algorithm of [11].

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.

2 Preliminaries

In this paper we consider simple undirected graphs. Let $G = (V, E)$ be a (weighted) 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 $N[D] = \cup_{v \in D} N[v]$. An *independent set* in G is a subset of pair-wise non-adjacent vertices. A *matching* is a subset of edges having no endpoints in common. 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. We refer to [1] for definitions of tree decomposition, path decomposition, treewidth and pathwidth of a graph. We denote by $\mathbf{tw}(G)$ and $\mathbf{pw}(G)$, treewidth and pathwidth of the graph G.

We need the following bounds on the pathwidth (treewidth) of sparse graphs. The proof of Lemma 1 is simple and based on the result of [5] and by induction on the number of vertices in a graph.

Lemma 1. *For any $\varepsilon > 0$, there exists an integer n_ε such that for every graph G with $n > n_\varepsilon$ vertices and $m = \beta n$ edges, $1.5 \leq \beta \leq 2$, the treewidth of G is bounded by $(m - n)/3 + \varepsilon n$.*

3 Minimum Maximal Matching

Given a graph $G = (V, E)$, any set of pairwise disjoint edges is called a *matching* of G. 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, that is Minimum Maximal Matching (MMM).

We need the following proposition which is a combination of two classical results due to Moon and Moser [10] and Johnson, Yannakakis and Papadimitriou in [8].

Proposition 1 ([8, 10]). *Every graph on n vertices contains at most $3^{n/3} = O(1.4423^n)$ maximal (with respect to inclusion) independent sets. Moreover, all these maximal independent sets can be enumerated with polynomial delay.*

Since for every $S \subseteq V$, S is a vertex cover of G if and only if $V \setminus S$ is an independent set of G, Proposition 1 can be used for enumerating minimal vertex covers as well.

Our algorithm also uses the following characterization of a MMM.

Algorithm MMM(G**)**
Data: A graph G.
Result: A minimum maximal matching of G or a path decomposition of G.
  return `findMMM`$(G, G, \emptyset)$

Function 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 of G subject to H and C or a path decomposition of G.

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

Fig. 1. Algorithm for Minimum Maximal Matching.

Proposition 2 ([12]). *Let $G = (V, E)$ be a graph and M be a minimum maximal matching of G. Let*

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

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 2, S does not need to be a *minimal* vertex cover.

The proof of the next lemma is based on standard dynamic programming on graphs of bounded treewidth, and we omit it.

Lemma 2. *There exists an algorithm to compute a minimum maximal matching of a graph G on n vertices in time $O(3^p n)$ when a path decomposition of width at most p is given.*

The algorithm of Figure 1 outputs either a path decomposition of the input graph G of reasonable width or a minimum maximal matching of G. The parameter G of Function findMMM corresponds always to the original input graph, H is a subgraph of G and C is a vertex cover of $G - V(H)$. The algorithm consists of three parts.

Branch. 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. The algorithm outputs a path decomposition using Lemma 3. Then the algorithm stops without backtracking. A minimum maximal matching can then be found using the pathwidth algorithm of Lemma 2.

Enumerate minimal vertex covers. 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 2 is used to find a minimum maximal matching of G.

The conditions under which these different parts are executed have been carefully chosen to optimize the overall running time of the algorithm, including the pathwidth algorithm of Lemma 2. Note that a path decomposition is computed at most once in an execution of the algorithm as findMMM stops right after outputting the path decomposition. Also note that the minimal vertex covers of H can only be enumerated in a leaf of the search tree corresponding to the recursive calls of the algorithm, as no recursive call is made in this part.

We define a *branch node* of the search tree of the algorithm to be a recursive call of the algorithm. Such a branch node is uniquely identified by the triple (G, H, C), that is the parameters of findMMM.

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

Proof. The correctness of the algorithm is clear from the above discussions. Here we give the running time for the algorithm.
Time Analysis: In the rest of the proof we upper bound the running time of this algorithm. It is essential to provide a good bound on the width of the produced path decomposition of G. The following lemma gives us the desired bounds on the pathwidth. Its proof is easy and is based on the bound on the pathwidth given in Lemma 1.

Lemma 3. *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* $\boldsymbol{pw}(H) + |C|$. *In particular,*
(*a*) *If* $\Delta(H) \leq 3$, *then* $\boldsymbol{pw}(G) \leq (\frac{1}{6} + \varepsilon)|V(H)| + |C|$ *for any* $\varepsilon > 0$.
(*b*) *If* $\Delta(H) \leq 2$, *then* $\boldsymbol{pw}(G) \leq |C| + 1$. *A path decomposition of the corresponding width can be found in polynomial time.*

Set $\alpha = 0.17385$ and $\beta = 0.31154$. First, consider the conditions under which a path decomposition may be computed. By combining the pathwidth bounds of Lemma 3 and the running time of the algorithm of Lemma 2, we obtain that MMM can be solved in time $O(max(3^{(1+5\alpha)/6}, 3^{\beta})^n)$ 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 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 $n = |V(G)|$ 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 $0 \leq r \leq k/4$. 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 $l \in L_r$ 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)**. So, 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

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

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 degree 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 1 to bound the running time of the enumerative step of the algorithm. Thus we obtain:

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

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

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

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

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 us 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 $0 \leq k \leq n$,

$\binom{n}{k} \leq \frac{(1+x)^n}{x^k}$. 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 $0.3090 < r < 0.3091$. Thus $c < 1.3091$. The value of d is computed in a similar way. □

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

$$O\left(max\left(3^{(1-\beta)/3}d^{\beta-\alpha}c^{\alpha}, 3^{(1+5\alpha)/6}, 3^{\beta}\right)^n\right) \leq O(1.4082^n) \ .$$

□

4 Counting Maximum Weighted Independent Sets

In this section we give an algorithm counting the number of maximum weighted independent sets in a graph, that is an algorithm for the #MWIS problem .

Most of the recent advances in counting maximum weighted independent sets are based on a reduction to counting satisfiable assignments of a 2-SAT formula. All these algorithms are based on the Branch & Reduce paradigm and involve detailed case distinctions. Here, we present a simple algorithm that combines Branch & Reduce and dynamic programming on graphs of bounded treewidth. It is well known (see for example [1]) that a maximum independent set can be found in time $O(2^{\ell}n)$ in a graph if a tree decomposition of width at most ℓ is given. This algorithm can be slightly modified to find the total number of maximum weighted independent sets in a graph with treewidth ℓ without increasing the running time of the algorithm.

Proposition 3. *Given a graph G with n vertices and a tree decomposition of G of width at most ℓ, all maximum weighted independent sets of G can be counted in time $O(2^{\ell}n)$.*

Our algorithm **#MWIS**, to count all maximum weighted independent sets of a graph, is depicted in Figure 2. The algorithm branches on a vertex v chosen by the following function which returns the vertex selected by the first applicable rule.

Pivot(G)

1. If $\exists v \in V$ such that $d(v) \geq 7$, then return v.
2. If $\exists v \in V$ such that $G - v$ is disconnected, then return v.
3. If $\exists u, v \in V$ such that $G - \{u, v\}$ is disconnected and $d(u) \leq d(v)$, then return v.
4. Return a vertex v of maximum degree such that $\sum_{u \in N(v)} d(u)$ is maximized.

When $|E| \leq 1.5185n$ the treewidth algorithm counts all maximum weighted independent sets. The procedure #MWISTW is a dynamic programming algorithm solving #MWIS of running time given in Proposition 3. When the algorithm branches on a vertex v, two subproblems are created according to the following rules:

(R1) add v to the partially constructed independent set and delete $N[v]$ from the graph.
(R2) delete v from the graph.

The correctness of the algorithm is clear from the presentation. Now we discuss its time complexity in detail which is based on a careful counting of subproblems of different size. We also need the following lemma for the analysis of the time complexity.

Algorithm #MWIS ($G = (V, E), w$)
Input: A graph $G = (V, E)$ and a weight function $w : V \to \mathbb{R}^+$.
Output: A couple $(size, nb)$ where $size$ is the maximum weight of an independent set of G and nb the number of different independent sets of this weight.

```
if G is disconnected with connected components G_1, ..., G_k then
    s' ← Σ_{i=1}^{k} s_i,  n' ← Π_{i=1}^{k} n_i where (s_i, n_i) ← #MWIS(G_i, w)
    return (s', n')
if |E| ≤ 1.5185|V| then
    return #MWISTW(G)
else
    v ← Pivot(G)
    (s_1, n_1) ← #MWIS(G − N[v], w)
    (s_2, n_2) ← #MWIS(G − {v}, w)
    s_1 ← s_1 + w(v)
    if s_1 > s_2 then
        return (s_1, n_1)
    else if s_1 = s_2 then
        return (s_1, n_1 + n_2)
    else
        return (s_2, n_2)
```

Fig. 2. An Algorithm to count all Maximum Weighted Independent Sets of a Graph

Lemma 4 ([3, 7]). *Let $G = (V, E)$ be a graph with n vertices, m edges, average degree $a > 0$ and v be a vertex chosen by Rule* 4 *of the function* **Pivot**. *Then the average degrees of $G - v$ and $G - N[v]$ are less than a.*

Theorem 2. *Let $G = (V, E)$ be a graph on n vertices.Algorithm* ***#MWIS*** *computes the number of* Maximum Weighted Independent Sets *of G in time $O(1.2431^n)$.*

Proof. (**1**) If G has at most $1.5n$ edges, its treewidth is at most $(1/6 + \epsilon)n$ by Lemma 1. Since only the treewidth algorithm is executed in this case, the running time is $O(2^{(1/6+\epsilon)n}) = O(1.1225^n)$.

(**2**) Assume that G has at most $2n$ edges. The worst case of the algorithm is when it branches on a vertex chosen by Rule 4 of the function **Pivot**. In this case the algorithm branches on vertices of degree at least 4 and executes the treewidth algorithm when $m \leq 1.5185n$. Set $\beta = 1.5185$. Let xn be the number of times the algorithm branches. The constant x is at least 0 and satisfies the inequality $2n - 4xn \geq \beta(n - xn)$ which implies that $x \leq (2 - \beta)/(4 - \beta)$.

Let s be the number of times the algorithm branches according to **(R1)**. Then, the algorithm branches $xn - s$ times according to **(R2)**. When the treewidth algorithm is executed, the size of the remaining graph is at most $n - 5s - (xn - s) = n - xn - 4s$. By Lemma 1, **tw**(G) is bounded by $n(\beta - 1)/3$. Let $T_{tw}(n) = 2^{n(\beta-1)/3}$ be the bound on the running time of the treewidth algorithm when executed on a graph with n vertices and βn edges. The running time of the algorithm is then bounded by

$$T_1(n) = \sum_{s=0}^{xn} \binom{xn}{s} T_{tw}(n - xn - 4s) = 2^{\frac{\beta-1}{3}(1-x)n}(1 + 2^{-4\frac{\beta-1}{3}})^{xn} \quad .$$

$T_1(n)$ is maximum when $x = (2-\beta)/(4-\beta)$. By replacing x by this value, we have that $T_1(n) \leq 1.20935^n$.

With an analysis similar to the one for the case when the graph has at most $2n$ edges, we can show that the Algorithm **#MWIS** takes $O(1.23724^n)$ time or $O(1.2431^n)$ time when the graph has at most $2.5n$ edges or $3n$ edges respectively.

(**3**) Now, if G has more than $3n$ edges then it contains vertices of degree at least 7 and hence the running time of the algorithm is smaller because the recurrence $T(n) = T(n-8) + T(n-1)$ solves to $O(1.2321^n)$.

□

5 Application to Parameterized Algorithms

Here we apply our technique to design a fixed parameter tractable algorithm for the parameterized version of Weighted Vertex Cover.

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 [11] that we use in our algorithm.

Proposition 4 ([11]). *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 $O(kn + k^3)$ either concludes that G has no vertex cover of weight $\leq k$, or outputs a kernel of size $\leq 2k$.*

Our algorithm is very similar to the one presented for counting all maximum independent sets. First we apply Proposition 4 to obtain a kernel of size at most $2k$. Then, as long as $|E| > 3.2k$, the algorithm branches on a vertex v chosen by the function **Pivot** as in the algorithm presented in Figure 2. If $|E| \leq 3.2k$, then by Lemma 1, a tree decomposition of small width (tw) can be found in polynomial time and we can use a $O(2^{tw}n)$ dynamic programming algorithm to solve k-Weighted Vertex Cover.

Theorem 3. *k-WVC on a graph on n vertices can be solved in time $O(1.3570^k n)$.*

6 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 #MWIS and MMM. Other problems for which we obtain faster algorithms using this technique include variants of MMM that are Minimum Edge Dominating Set and Matrix Domination Set. We also applied the technique to design fixed parameter tractable algorithms and obtained the best known algorithm for k-WVC 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 highly *non-trivial*. Our improvement in the runtime for #MWIS directly gives improved algorithms for #1-IN-k-SAT, #Exact Hitting Set, #Exact Cover and

#Weighted Set Cover. The definitions and reductions of these problems to #MWIS can be found in [2]. Other parameterized problems for which we obtain the fastest known algorithms using the techniques developed in this paper are the *weighted* and *unweighted* version of *parameterized minimum maximal matching* and *minimum edge dominating set*, which will appear in the longer version of this paper.

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. H. L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theoretical Computer Science, 209 (1998), pp. 1–45.
2. 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.
3. V. Dahllöf, P. Jonsson, and M. Wahlström, *Counting models for 2SAT and 3SAT formulae*, Theoretical Computer Science, 332 (2005), pp. 265–291.
4. F. V. Fomin, F. Grandoni, and D. Kratsch, *Some new techniques in design and analysis of exact (exponential) algorithms*, Bulletin of the EATCS, 87 (2005), pp. 47–77.
5. F. V. Fomin and K. Høie, *Pathwidth of cubic graphs and exact algorithms*, Information Processing Letters, 97 (2006), pp. 191–196.
6. M. Fürer and S. P. Kasiviswanathan, *Algorithms for counting 2-SAT solutions and colorings with applications*, Electronic Colloquium on Computational Complexity (ECCC), vol. 33, 2005.
7. M. K. Goldberg, D. Berque, and T. Spencer *A Low-Exponential Algorithm for Counting Vertex Covers*, Graph Theory, Combinatorics, Algorithms, and Applications, vol. 1, (1995), pp. 431–444.
8. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, *On generating all maximal independent sets*, Information Processing Letters, 27 (1988), pp. 119–123.
9. J. Kneis, D. Mölle, S. Richter, and P. Rossmanith, *Algorithms based in treewidth of sparse graphs*, in Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005), vol. 3787 of LNCS, Springer, (2005), pp. 385–396.
10. J. W. Moon and L. Moser, *On cliques in graphs*, Israel Journal of Mathematics, 3 (1965), pp. 23–28.
11. R. Niedermeier and P. Rossmanith, *On efficient fixed-parameter algorithms for weighted vertex cover*, Journal of Algorithms, 47 (2003), pp. 63–77.
12. V. Raman, S. Saurabh, and S. Sikdar, *Efficient exact algorithms through enumerating maximal independent sets and other techniques*, Theory of Computing Systems, to appear.
13. B. Randerath and I. Schiermeyer, *Exact algorithms for MINIMUM DOMINATING SET*, Technical Report zaik-469, Zentrum für Angewandte Informatik Köln, Germany, 2004.
14. G. Woeginger, *Exact algorithms for NP-hard problems: A survey*, in Combinatorial Optimization - Eureka, you shrink!, vol. 2570 of LNCS, Springer, (2003), pp. 185–207.