Inclusion/Exclusion Meets Measure and Conquer

Exact Algorithms for Counting Dominating Sets

Johan M.M. van Rooij[1], Jesper Nederlof[2,⋆], and Thomas C. van Dijk[1]

[1] Department of Information and Computing Sciences, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands
{jmmrooij,thomasd}@cs.uu.nl
[2] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
Jesper.Nederlof@ii.uib.no

Abstract. In this paper, two central techniques from the field of exponential time algorithms are combined for the first time: inclusion/exclusion and branching with measure and conquer analysis.

In this way, we have obtained an algorithm that, for each κ, counts the number of dominating sets of size κ in $\mathcal{O}(1.5048^n)$ time. This algorithm improves the previously fastest algorithm that counts the number of minimum dominating sets. The algorithm is even slightly faster than the previous fastest algorithm for minimum dominating set, thus improving this result while computing much more information.

When restricted to c-dense graphs, circle graphs, 4-chordal graphs or weakly chordal graphs, our combination of branching with inclusion/exclusion leads to significantly faster counting and decision algorithms than the previously fastest algorithms for dominating set.

All results can be extended to counting (minimum) weight dominating sets when the size of the set of possible weight sums is polynomially bounded.

1 Introduction

Recently, the field of exact exponential time algorithms has been an area of growing interest. Maybe the most notable recent developments are *measure and conquer* [10,11] and *inclusion/exclusion* [1,5,17]. Both techniques have been demonstrated on Set Cover problems in early stages: measure and conquer was introduced on a set cover formulation of Dominating Set, and in [5] inclusion/exclusion was used for counting sets coverings and set partitionings.

In this paper, we show that it is possible to use both techniques in one combined approach. This allows for fast measure and conquer running times on inclusion/exclusion based algorithms.

⋆ This research was done while all authors were associated with Utrecht University.

A. Fiat and P. Sanders (Eds.): ESA 2009, LNCS 5757, pp. 554–565, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The best known shape of inclusion/exclusion is the formula summing over some powerset; see [3,5,17]. However, the fundamental branching perspective from [1] is more direct and powerful. In this paper, we will apply this branching perspective to set cover instances obtained from the set cover formulation of dominating set that has been used to introduce measure and conquer [10,16].

In this setting, we use a traditional branching rule to branch on a set, or an application of inclusion/exclusion to branch on an element. The sole application of either one of these strategies gives a typical exhaustive search or the aforementioned shape of inclusion/exclusion sum, respectively. We use both branching strategies in unity obtaining a *mixed inclusion/exclusion branching* algorithm that can be analysed using measure and conquer.

Until 2004, no exact algorithm for Dominating Set beating the trivial $\mathcal{O}(2^n n^{\mathcal{O}(1)})$ was known. In that year, three algorithms were published [13,16,20], the fastest of which is Grandoni's running in time $\mathcal{O}(1.8019^n)$ [16]. One year later, the algorithm of Grandoni was analysed using measure and conquer giving a bound of $\mathcal{O}(1.5137^n)$ on the running time [10]. This was later improved by Van Rooij and Bodlaender [21] to $\mathcal{O}(1.5063^n)$.

When we want to *count minimum dominating sets*, there is an algorithm by Fomin et al. running in time $\mathcal{O}(1.5535^n)$ [9]. This algorithm combines branching with path decomposition techniques: something we will use for our own algorithm as well. Also related is a result by Björklund and Husfeldt solving this problem on cubic graphs in $\mathcal{O}(1.3161^n)$ using path decompositions in combination with inclusion/exclusion [3]. To our knowledge, there are no existing algorithms combining measure and conquer with inclusion/exclusion.

Our algorithm is more general. It counts the number of dominating sets in an n-vertex graph of each size $0 \leq \kappa \leq n$, with an upper bound on the running time of $\mathcal{O}(1.5048^n)$. This is slightly faster than even the current fastest algorithm that computes a minimum dominating set.

Gaspers et al. [14] show that algorithms for the set cover formulation of dominating set can be combined with dynamic programming over tree decompositions to obtain faster running times for the dominating set problem restricted to some graph classes. These classes are c-dense graphs, chordal graphs, circle graphs, 4-chordal graphs and weakly chordal graphs. We show that our mixed branching approach with inclusion/exclusion branches works even better on four of these graph classes; we do not only improve these results because we have a faster algorithm for the underlying set cover problem, but do so more significantly by exploiting vertices of high degree twice by using both techniques. Moreover, we can count the number of dominating sets of each size, in contrast to the previous results that only compute a single minimum dominating set.

2 Preliminaries

We consider the #κ-Dominating Set problem: how many dominating sets of size κ exist for G, i.e., how many subsets $V' \subseteq V$ with $|V'| = \kappa$ such that for all $u \in V \backslash V'$ there is a $v \in V'$ for which $(u, v) \in E$?

We formulate the problem as the set cover variant #κ-Set Cover [16]: given a collection of subsets $\mathcal{S}$ of a finite universe $\mathcal{U}$ and a positive integer κ, how many set covers for $\mathcal{U}$ of size κ does $\mathcal{S}$ contain? The transformation between this problem and our original problem is straightforward: for every vertex in $v \in V$ we introduce both an element in $\mathcal{U}$ and a set in $\mathcal{S}$ corresponding to $N[v]$. We now use the *cardinality* $|S|$ of the set S and the *frequency* $f(e)$ of the element e instead of the degree of a vertex. The *dimension* of a set cover instance is defined as $\dim(\mathcal{S},\mathcal{U}) = |\mathcal{S}|+|\mathcal{U}|$. Hence, an n-vertex dominating set instance is transformed into a set cover instance of dimension $d = 2n$.

We also look at the problem as a #κ-Red/Blue Dominating Set problem in the incidence graph of the set cover instance [9]. The *incidence graph* is the bipartite graph with *red vertices* $V_{Red} = \mathcal{S}$ and *blue vertices* $V_{Blue} = \mathcal{U}$. Vertices $S \in V_{Red}$ and $u \in V_{Blue}$ are adjacent if and only if $u \in S$. In this problem, we count the number of ways to take κ red vertices to dominate all the blue vertices. It is easy to see that this perspective is equivalent to the set cover variant.

Finally, we assume the reader to be familiar with the concepts of a (nice) tree decomposition and a (nice) path decompositions of a graph, and how to perform dynamic programming over these structures. For a good overview see [6,7].

3 Inclusion/Exclusion Based Branching

We will begin by showing that one can look at Inclusion/Exclusion as a branching rule [2]. In this way, we can Inclusion/Exclusion-branch on an element in a Set Cover instance in the same way as one would normally branch on a set.

A set S is *optional* in an instance, if either S is in the solution, or S is not. Branching on this choice is straightforward: the total number of set covers of size κ equals the number of set covers of size $\kappa - 1$ after we take S (*require*), plus the number of set covers of size κ after we discard S (*forbid*). I.e.:

$$\text{OPTIONAL} = \text{REQUIRED} + \text{FORBIDDEN}$$

We now consider branching on an element [2]. This may appear strange at first as elements are not optional. Inspired by Inclusion/Exclusion techniques, we can, however, rearrange the above formula to give:

$$\text{REQUIRED} = \text{OPTIONAL} - \text{FORBIDDEN}$$

That is, the number of solutions that cover an element e is equal to the number of solutions in which covering e is *optional* (maybe cover e), minus the number of solutions in which covering e is *forbidden* (that do not cover e). We call this type of branching *inclusion/exclusion based branching* or simply *IE-branching*.

Notice that both branching rules are symmetric when applied to the incidence graph representation of our problem: in one branch a (red or blue) vertex is removed, and in the other, this vertex and its neighbours are removed.

An algorithm that branches on every set is called *exhaustive search*, while an algorithm that solely use IE-branching is an *inclusion/exclusion* algorithm.

To see the relation to the inclusion/exclusion formula [5], let c_κ be the number of set covers of cardinality κ, and let $a(X)$ be the number of sets in $\mathcal{S}$ that do not contain any element in X. Consider the branching tree after exhaustively applying IE-branching and look at the contribution of a leaf to the total number computed. In each leaf of the tree, each element is either optional or forbidden; the $2^{|\mathcal{U}|}$ leaves represent the possible subsets $X \subseteq \mathcal{U}$ of forbidden elements. The contribution of this leaf equals the number of set covers of cardinality κ where it is optional to cover each element not in X and forbidden to cover an element in X, i.e. $\binom{a(X)}{\kappa}$. A minus sign is added for each time we have entered a forbidden branch, so the total contribution equals $(-1)^{|X|}\binom{a(X)}{\kappa}$. This leads to the formula given below on the left. Compare this to the inclusion/exclusion formula [5] on the right: the difference comes from the fact that Björklund et al. allow a single set to be picked multiple times while we do not.

$$c_\kappa = \sum_{X \subseteq \mathcal{U}} (-1)^{|X|} \binom{a(X)}{\kappa} \qquad\qquad c'_\kappa = \sum_{X \subseteq \mathcal{U}} (-1)^{|X|} a(X)^\kappa$$

4 An Algorithm for Counting Dominating Sets

We start by applying our combined technique to the problem of counting dominating sets. The previously fastest algorithm that counts the number of *minimum* dominating sets is by Fomin et al. [9]. Their algorithm combines pathwidth techniques with branching and measure and conquer analysis. We present a modification of this algorithm (Algorithm 1) that solves the #κ-Dominating Set problem. This algorithm computes the number of dominating sets of each size κ ($0 \leq \kappa \leq n$) in $O(1.5048^n)$ time improving both the results of [9] and the previously fastest algorithm for minimum dominating set [21].

Algorithm 1 works on the #κ-Set Cover transformation of the problem and returns a list containing the number of set covers $\boldsymbol{n}_\kappa$ of size κ for each $0 \leq \kappa \leq n$. It is a branch and reduce algorithm, branching both on sets and elements as discussed in Section 3. When an instance generated by the branching is sparse enough, the algorithm will compute a path decomposition of the incidence graph of the instance. The algorithm then solves this instance by dynamic programming on this path decomposition.

Algorithm 1 takes as input a collection of sets $\mathcal{S}$ forming a #κ-Set Cover instance, and a multiplicity function m. This function m exists because we want to avoid identical sets to be created by the algorithm; the algorithm deals with multiple identical sets by using the multiplicity counters in m. We will begin by describing a series of polynomial time reduction rules that Algorithm 1 applies before branching or applying pathwidth techniques.

Base Case
Some inputs can be completely reduced to a collection of m' empty sets by the reduction rules below. There are no elements left, and we only have empty sets to choose from, therefore the algorithm returns $\boldsymbol{n}_\kappa = \binom{m'}{k}$ for each $0 \leq \kappa \leq n$.

Algorithm 1. `Count-SC(`$\mathcal{S}$`,`m`)`

Input: A collection of sets $\mathcal{S}$ over the universe $\mathcal{U} = \cup\mathcal{S}$ and a multiplicity function m.
Output: A list $\boldsymbol{n}$ of length $\#(\mathcal{S}) + 1$ containing the number of set covers of $(\mathcal{S},\mathcal{U})$ of each size $0 \leq \kappa \leq \#(\mathcal{S})$.

//reduction rules
if $\mathcal{S}$ equals a collection of $m' = m(\emptyset)$ empty sets **then** //base case
  return $(\binom{m'}{0}, \binom{m'}{1}, \ldots, \binom{m'}{m'})$
else if there exist identical sets in $\mathcal{S}$ **then** //identical sets
  Remove the identical sets from $\mathcal{S}$ and update the multiplicity function m.
  return `Count-SC(`$\mathcal{S}$`,`m`)`
else if there exists an element $e \in \mathcal{U}$ of frequency one **then** //unique elements
  Let $\mathcal{S}'$ be $\mathcal{S}$ after removing the set S with $e \in S$ and let $\boldsymbol{n}_{\text{take}} =$ `Count-SC(`$\mathcal{S}$`,`m`)`
  return $\boldsymbol{n}_{\text{take}}$ after updating it using the multiplicity of S in formula 1
else if there exist two elements $e, e' \in \mathcal{U}$ such that for all $S \in \mathcal{S}$: if $e \in S$, then $e' \in S$ **then** //subsumption
  Let $\mathcal{S}' = \{S \backslash \{e'\} \mid S \in \mathcal{S}\}$, and update m such that it now works on $\mathcal{S}'$
  return `Count-SC(`$\mathcal{S}'$`,`m`)`
else if if $\mathcal{S}$ can be partitioned in two subcollections $\mathcal{C}, \bar{\mathcal{C}}$ such that every element of $e \in \mathcal{U}$ occurs either in $\mathcal{C}$ or in $\bar{\mathcal{C}}$ and not in both **then** //connected components
  Let $\boldsymbol{n}_{\mathcal{C}} =$ `Count-SC(`$\mathcal{C}$`,`m`)`, and $\boldsymbol{n}_{\bar{\mathcal{C}}} =$ `Count-SC(`$\bar{\mathcal{C}}$`,`m`)`
  return The solution to $\mathcal{S}$ by merging $\boldsymbol{n}_{\mathcal{C}}$ and $\boldsymbol{n}_{\bar{\mathcal{C}}}$ using formula 2
end if

//branching or path decomposition
Let $S \in \mathcal{S}$ be of maximum cardinality and not an exceptional case[1]
Let $e \in \mathcal{U}$ be of maximum frequency, also not an exceptional case[1]
Preference order P: $\mathsf{S}_4 < \mathsf{S}_5 < \mathsf{S}_6 < \mathsf{E}_5 < \mathsf{E}_6 < \mathsf{S}_7 < \mathsf{E}_7 < \mathsf{E}_{\geq 8} = S_{\geq 8}$
if $\mathsf{S}_{|S|}$ and $\mathsf{E}_{\text{freq}(e)}$ are too small to be in P **then** //path decomposition
  Compute a path decomposition P_I of the incidence graph of $(\mathcal{S},\mathcal{U})$
  return The solution to $\mathcal{S}$ obtained by dynamic programming over P_I.
else if $\mathsf{E}_{\text{freq}(e)}$ is in the order P and $\mathsf{E}_{\text{freq}(e)} \not< \mathsf{S}_{|S|}$ **then** //element branch
  Let $\mathcal{S}' = \{S \backslash \{e'\} \mid S \in \mathcal{S}\}$, and update m such that it now works on $\mathcal{S}'$
  Let $\boldsymbol{n}_{\text{optional}} =$ `Count-SC(`$\mathcal{S}'$`,`m`)`
  Let $\boldsymbol{n}_{\text{forbidden}} =$ `Count-SC(`$\mathcal{S} \setminus \{S \in \mathcal{S} \mid e \in S\}$`,`$m$`)`
  return $\boldsymbol{n}_{\text{optional}} - \boldsymbol{n}_{\text{forbidden}}$
else //$\mathsf{S}_{|S|}$ is in the order P and $\mathsf{S}_{|S|} \not< \mathsf{E}_{\text{freq}(e)}$ //set branch
  Let $\mathcal{S}' = \{S' \backslash S \mid S' \in (\mathcal{S} \setminus \{S\})\}$, and update m such that it now works on $\mathcal{S}'$
  Let $\boldsymbol{n}_{\text{take}} =$ `Count-SC(`$\mathcal{S}'$`,`m`)`
  Update $\boldsymbol{n}_{\text{take}}$ using the multiplicity of S in formula 1
  Let $\boldsymbol{n}_{\text{discard}} =$ `Count-SC(`$\mathcal{S} \setminus \{S\}$`,`$m$`)`
  return $\boldsymbol{n}_{\text{take}} + \boldsymbol{n}_{\text{discard}}$
end if

[1] There are some exceptional combinations of cardinalities of sets and frequencies of elements on which the algorithm will not branch. These will be handled by the path decomposition phase. For a complete list of these cases see Overview 1.

Identical Sets
When $\mathcal{S}$ contains identical sets, we remove all but one copies of this set and keep track of this using multiplicity counters in m. We can do this because taking at least one copy in a solution will result in the same subproblem regardless of the number of copies chosen. Whenever the set is explicitly taken in a solution later on, we compute the required result from the values from the recursive call n'_κ using the formula below.

$$\boldsymbol{n}_\kappa = \sum_{i=1}^{m} \binom{m}{i} \boldsymbol{n}'_{\kappa-i} \tag{1}$$

To avoid confusion, we consider copies of sets to be removed when considering the frequency of its elements or the number of sets in $\mathcal{S}$.

Unique Elements
Whenever there exists an element e of frequency one in $\mathcal{U}$, the set S containing e must belong to every set cover. Therefore, the algorithm acts as if it takes this set and goes in recursion on the instance with S and all its elements removed, counting the number of set covers of size κ-1. Notice that it is not a problem if the set taken has multiplicity greater than one: simply use the above formula.

Subsumption
If there exists an element e which occurs in every set (and possibly more) in which another element e' occurs, then every set cover that covers e also covers e'. Thus, we can remove e' from the instance and recursively apply our algorithm.

Connected Components
If the incidence graph of the instance contains multiple connected components, then we can solve the problem on each component separately and merge the results. In this case, there exist two disjoint sets $\mathcal{C}$, $\bar{\mathcal{C}}$ with $\mathcal{C} \cup \bar{\mathcal{C}} = \mathcal{S}$ and with the property that every element of $e \in \mathcal{U}$ occurs either in $\mathcal{C}$ or in $\bar{\mathcal{C}}$ and not in both. Let $\boldsymbol{n}(\mathcal{C})_\kappa$, $\boldsymbol{n}(\bar{\mathcal{C}})_\kappa$ be the number of solutions of size κ to these two subproblems. In order to compute the total number of size covers n_κ of size κ in $\mathcal{C} \cup \bar{\mathcal{C}}$ we evaluate the following sum:

$$\boldsymbol{n}_\kappa = \sum_{i=0}^{\kappa} \boldsymbol{n}(\mathcal{C})_i \times \boldsymbol{n}(\bar{\mathcal{C}})_{\kappa-i} \quad \text{where: } \boldsymbol{n}(\mathcal{C})_i = 0 \text{ if } i > |\mathcal{C}| \tag{2}$$

Branching
When no reduction rules are applicable, the algorithm chooses a set or an element to branch on. From the instance, it chooses a set of maximum cardinality and an element of maximum frequency that are both not exceptional cases. We postpone the discussion of these exceptional cases for a moment. In order to choose between branching on the chosen set and branching on the chosen element, the algorithm uses the following preference order P:

$$\mathsf{P}: \quad \mathsf{S}_4 < \mathsf{S}_5 < \mathsf{S}_6 < \mathsf{E}_5 < \mathsf{E}_6 < \mathsf{S}_7 < \mathsf{E}_7 < \mathsf{E}_{\geq 8} = S_{\geq 8}$$

There are exceptional cases of elements on which, despite the preference order, our algorithm does not branch. These cases represent local neighbourhoods of sets or elements which would increase the running time of the algorithm when branched on, but can be handled by dynamic programming on a path decomposition quite effectively. The exceptional cases are:

1. Elements of frequency five that occur in many sets of small cardinality.
   Let the 5-tuple $(s_1, s_2, s_3, s_4, s_5, s_6)$ represent a frequency five element occurring s_i times in a cardinality i set. In this way, our special cases can be denoted as:

(1, 4, 0, 0, 0, 0) - (0, 5, 0, 0, 0, 0) - (1, 3, 1, 0, 0, 0) - (0, 4, 1, 0, 0, 0) - (1, 2, 2, 0, 0, 0)
(0, 3, 2, 0, 0, 0) - (1, 1, 3, 0, 0, 0) - (0, 2, 3, 0, 0, 0) - (0, 1, 4, 0, 0, 0) - (1, 0, 4, 0, 0, 0)
(1, 3, 0, 1, 0, 0) - (0, 4, 0, 1, 0, 0) - (1, 2, 1, 1, 0, 0) - (0, 3, 1, 1, 0, 0) - (1, 1, 2, 1, 0, 0)
(1, 0, 3, 1, 0, 0) - (1, 2, 0, 2, 0, 0) - (1, 3, 0, 0, 1, 0) - (1, 2, 1, 0, 1, 0) - (1, 3, 0, 0, 0, 1)

2. Sets of cardinality four, five or six, containing one of the elements described above.

Overview 1. Exceptional cases for our algorithm

In this ordering, $\mathsf{S}_i < \mathsf{E}_j$ means that the algorithm prefers to branch on an element of frequency j over branching on a set of cardinality i.

Sets of cardinality at most three and elements of frequency at most four do not occur in the preference order P. These are considered too small for efficient branching since branching on them would remove or reduce too few elements and sets. The remaining instances are handled by dynamic programming over a path decomposition of the incidence graph, similar to [9].

The exceptional cases are described in Overview 1. These exceptional cases represent local neighbourhoods around a set or an element which, despite the general rule imposed by the preference order, can be handled more efficiently by the path decomposition phase of our algorithm than by branching. They exist to properly balance the two parts of the algorithm.

Theorem 1. *There is an algorithm that solves the* #κ-Dominating Set *for all* $0 \leq \kappa \leq n$ *in an* n*-vertex graph* G *in* $\mathcal{O}(1.5048^n)$.

Proof (Sketch). The proof consists of a measure and conquer analysis of Algorithm 1 and is an extension of the proof in [9]. Due to space restrictions, we will only sketch it here. For the full proof, see [23].

We analyse our algorithm using measure and conquer [10,11]; see also [15,21]. Let $v, w : \mathbb{N} \to [0,1]$ be weight functions giving weight $v(i)$ to an element of frequency i and weight $w(i)$ to a set of cardinality i, respectively. With these functions we define the following complexity measure (identical to [10,21]):

$$k(\mathcal{S},\mathcal{U}) = \sum_{S \in \mathcal{S}} w(|S|) + \sum_{e \in \mathcal{U}} v(f(e)) \qquad \text{notice: } k(\mathcal{S},\mathcal{U}) \leq \dim(\mathcal{S},\mathcal{U})$$

We derive recurrence relations for the number of subproblems generated by the branching of the algorithm expressed in this complexity measure k. Given fuctions v, w, we can solve these recurrences and obtain an upper bound on the

number of subproblems generated. The proof the comes down to computing the functions v, w that minimise the running time. This is a quasiconvex program [8] that we solve by computer. In this way, we prove:

Let $N_h(k)$ be the number of subproblem of complexity h generated by our algorithm on an input of complexity k. Then, $N_h(k) < 1.22670^{k-h}$.

Next, we use that by standard dynamic programming techniques we can solve the problem on a path decomposition of width p in $\mathcal{O}^*(2^p)$. We compute an upper bound on the maximum width p any path decomposition that is computed by our algorithm can have. Using upper bound on the pathwidth of sparse graphs [9,12], we formulate a linear program that computes the maximum pathwidth that any instance of complexity k. As a result we find that $p < 0.28991k$, and thus this part of the algorithm runs in $\mathcal{O}(2^{0.28991k}) \subset \mathcal{O}(1.2226^k)$.

By combining the time bound on both parts of the algorithm and using that initially $k \leq 2n$, we conclude that Algorithm 1 runs in $\mathcal{O}(1.22670^{2n}) \subset \mathcal{O}(1.5048^n)$. □

5 Dominating Set Restricted to Some Graph Classes

The algorithm from the previous section, not only gives the currently fastest algorithm to compute the number of dominating sets of given sizes, but also is the currently fastest algorithm for the minimum dominating set problem. However, the improvement over the previous fastest minimum dominating set algorithm [21] is small. When we consider the dominating set problem on specific graph classes, we get a larger improvement with our approach. This also extends the results on these graph classes to the counting variant of Dominating Set.

Gaspers et al. [14] consider exact algorithms for the dominating set problem on c-dense graphs, circle graphs, chordal graphs, 4-chordal graphs, and weakly chordal graphs. On these graph classes the problem is still NP-complete. They show that if we restrict ourselves to such a graph class, then there are either many vertices of high degree allowing more efficient branching, or the graph has low treewidth allowing the problem to be efficiently solved by dynamic programming over a tree decomposition. Using our approach, we will show that less vertices of high degree are required to obtain the same effect by branching on them with both branching rules. This leads to faster algorithms.

If we combine the results of the previous section with a result from Gaspers el al. [14], then we have the following proposition:

Proposition 1 ([14], Theorem 1). *Let $t > 0$ be a fixed integer, and let $\mathcal{G}_t$ be a class of graphs with for all $G \in \mathcal{G}_t$: $|\{v \in V(G) \ : \ d(v) \geq t-2\}| \geq t$. Then, there is a $\mathcal{O}(1.22670^{2n-t})$ time algorithm to solve the minimum dominating set problem on graphs in $\mathcal{G}_t$.*

Using our two branching rules, we prove a stronger variant of this proposition.

Lemma 1. *Let $t > 0$ be a fixed integer, and let $\mathcal{G}_t$ be a class of graphs with for all $G \in \mathcal{G}_t$: $|\{v \in V(G) \ : \ d(v) \geq t-2\}| \geq \frac{1}{2}t$. Then, there is a $\mathcal{O}(1.22670^{2n-t})$ time algorithm to solve the minimum dominating set problem on graphs in $\mathcal{G}_t$.*

Table 1. Effect of two branching rules on the running times on some graph classes

Graph class	[14] (+[21]+[22])	[14] (+[22]) + Lemma 1
c-dense graphs	$\mathcal{O}\left(1.2273^{(1+\sqrt{1-2c})n}\right)$	$\mathcal{O}\left(1.2267^{(\frac{1}{2}+\frac{1}{2}\sqrt{9-16c})n}\right)$
circle graphs	$\mathcal{O}(1.4843^n)$	$\mathcal{O}(1.4806^n)$
chordal graphs	$\mathcal{O}(1.3720^n)$	$\mathcal{O}(1.3712^n)$ *
4-chordal graphs	$\mathcal{O}(1.4791^n)$	$\mathcal{O}(1.4741^n)$
weakly chordal graphs	$\mathcal{O}(1.4706^n)$	$\mathcal{O}(1.4629^n)$

* This result does not use Lemma 1; the improvement comes only from Theorem 1.

Proof. Let H be the set of vertices of degree at least $t-2$ from the statement of the lemma, and consider the set cover formulation of the dominating set problem.

Let S be a set corresponding to a vertex in H. We branch on this set and consider the branch in which we take this set in the set cover: the set is removed and all its elements are covered and hence removed also. These are at least $t-1$ elements, and therefore this branch results in a problem of dimension at most $2n-t$. Only a single set is removed in the other branch, in which case we repeat this process and branch on the next set represented by another vertex in H. This gives us $\frac{1}{2}t$ problem instances of dimension at most $2n-t$ and one problem instance of dimension $2n-\frac{1}{2}t$ because here $\frac{1}{2}t$ sets are removed.

In this latter instance, we use our new inclusion/exclusion based branching rule on the elements corresponding to the vertices in H. These elements still have frequency at least $\frac{1}{2}t-1$, since only $\frac{1}{2}t$ sets have been discarded until now. When branching on an element and forbidding it, a subproblem of dimension at most $2n-t$ is created because at least an additional element and $\frac{1}{2}t-1$ sets are removed in this branch. What remains is one subproblem generated in the branch after discarding $\frac{1}{2}t$ sets and making $\frac{1}{2}t$ elements optional. Since all these sets and elements are removed in these branches, this also gives us a problem of dimension $2n-t$.

The above procedure generates $t+1$ problems of dimension $2n-t$, which can all be solved by Algorithm 1 in $\mathcal{O}(1.22670^{2n-t})$ time. These are only a linear number of instances giving us a total running time of $\mathcal{O}(1.22670^{2n-t})$. □

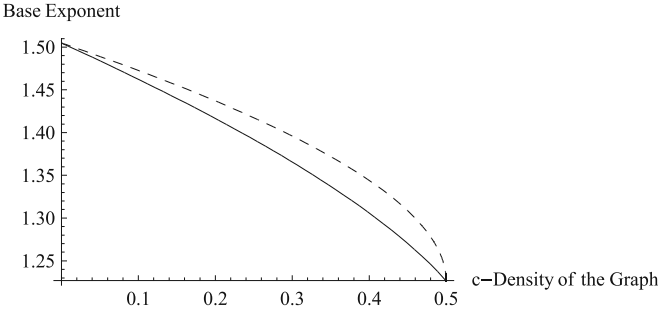
Using Lemma 1 and following the computations in [14] we have obtained the following result. Due to space restrictions we refer to [23] for the full proof.

Theorem 2. *There exist algorithms that count the number of dominating sets of each size in a c-dense graph in $\mathcal{O}\left(1.2267^{(\frac{1}{2}+\frac{1}{2}\sqrt{9-16c})n}\right)$ time, in a circle graph in $\mathcal{O}(1.4806^n)$ time, in a 4-chordal graph in $\mathcal{O}(1.4741^n)$ time, and in a weakly chordal graph in $\mathcal{O}(1.4629^n)$ time.*

See Figure 1 and Table 1 for a comparison of our results with [14].

6 Further Applications

In principle, we could take any inclusion/exclusion algorithm and reformulate it into a branching algorithm. Then, we can look for reduction rules transforming



The solid line represents the upper bound on the running time of our algorithm, and the dashed line represents the upper bound obtained from [14] after plugging in our faster algorithm for dominating set.

Fig. 1. Comparison of bounds on the running time on c-dense graphs

it into a branch and reduce algorithm. Finding these reduction rules is often not very hard. However, finding any reduction rules for which you can prove that it improves the worst case behaviour of the algorithm is often very hard.

For example, consider the problem of counting the number of perfect matchings in a graph. This is easily modified into a $\#(n/2)$-Set Cover instance with n elements and possibly $O(n^2)$ sets to which we can apply a branching algorithm using the reduction rules of Algorithm 1. However, for such an algorithm, it is of no use to branch on a set since their cardinalities are too small, and we obtain an $\mathcal{O}^*(2^n)$ algorithm using polynomial space as in [3].

What this approach does accomplish, is that a branch and reduce inclusion/exclusion algorithm no longer has the property that its worst and best case behaviour conincide. When using the inclusion/exclusion formula one always evaluates every term of the sum, while if we are branching, then the number of leaves of the search tree can very well be a lot smaller due to the reduction rules.

To apply our combined approach, branching both on sets and elements, we need to consider problems that can be transformed into variations of set cover instances having a linear number of sets and elements. In search of such problems, we, very recently, obtained several results when considering the problem of whether there exists a locally subjective homomorphism form a fixed graph H into the input graph G (also known as role-assignment problems) [19].

7 Conclusion

While the improvements of the running times for the studied problems are interesting, we believe that the most important contribution of our paper is the novel combination of inclusion/exclusion and branching with a measure and conquer analysis. This gives a nice way to create inclusion/exclusion algorithms without the usual $\mathcal{O}(2^n n^{\mathcal{O}(1)})$ running time.

Many counting and decision variants of dominating set can be translated to set cover problems and solved by our algorithms in the same time. For example: directed dominating set, total dominating set[2], k-distance dominating set[2], weak/strong dominating, and combinations of these. We can also solve the weighted versions of all these problems as long as the size of the set of possible weight sums Σ is polynomially bounded: modify the algorithm such that it computes the number of set covers of each possible weight $w \in \Sigma$ at each step.

Our running times are highly dependent on the current best known upper bounds on the pathwidth of bounded degree graphs [9,12]. Any result that would improve these bounds would also improve our algorithm.

We use path decompositions while tree decompositions are more general and allow the same running times when using fast subset convolutions [4] to perform join operations [22]. We consider it to be an important open problem to give stronger (or even tight) bounds on the treewidth [18] or pathwidth of bounded degree graphs for which decompositions can be computed efficiently.

Acknowledgements

We would like to thank our advisor Hans L. Bodlaender for his enthusiasm for this research and for useful comments on an earlier draft of this paper. We would also like to thank Alexey A. Stepanov for useful discussions on [9].

References

1. Bax, E.T.: Inclusion and exclusion algorithm for the hamiltonian path problem. Information Processing Letters 47(4), 203–207 (1993)
2. Bax, E.T.: Recurrence-based reductions for inclusion and exclusion algorithms applied to #P problems (1996)
3. Björklund, A., Husfeldt, T.: Exact algorithms for exact satisfiability and number of perfect matchings. Algorithmica 52(2), 226–249 (2008)
4. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: fast subset convolution. In: STOC 2007: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, pp. 67–74. ACM, New York (2007)
5. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM Journal of Computing, Special Issue for FOCS 2006 (to appear)
6. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica 11, 1–23 (1993)
7. Bodlaender, H.L., Koster, A.M.C.A.: Combinatorial optimization on graphs of bounded treewidth. The Computer Journal 51(3), 255–269 (2008)
8. Eppstein, D.: Quasiconvex analysis of backtracking algorithms. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, pp. 781–790 (2004)

[2] Total dominating set requires a $\mathcal{O}(4^t n^{\mathcal{O}(1)})$ algorithm on tree decompositions [22]. k-distance dominating set can also not be solved on tree decompositions in $\mathcal{O}(3^t n^{\mathcal{O}(1)})$ if $k > 1$. Hence the results from Section 5 do not extend to these problems.

9. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On two techniques of combining branching and treewidth. In: Algorithmica Special issue of ISAAC 2006 (2006) (to appear)
10. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: Domination — a case study. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 191–203. Springer, Heidelberg (2005)
11. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, pp. 18–25 (2006)
12. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Information Processing Letters 97, 191–196 (2006)
13. Fomin, F.V., Kratsch, D., Woeginger, G.J.: Exact (exponential) algorithms for the dominating set problem. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 245–256. Springer, Heidelberg (2004)
14. Gaspers, S., Kratsch, D., Liedloff, M.: Exponential time algorithms for the minimum dominating set problem on some graph classes. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 148–159. Springer, Heidelberg (2006)
15. Gaspers, S., Sorkin, G.B.: A universally fastest algorithm for max 2-sat, max 2-csp, and everything in between. In: Proceedings of the 20th annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009). SIAM, Philadelphia (to appear, 2009)
16. Grandoni, F.: A note on the complexity of minimum dominating set. J. Disc. Alg. 4, 209–214 (2006)
17. Karp, R.M.: Dynamic programming meets the principle of inclusion-exclusion. Operations Research Letters 1, 49–51 (1982)
18. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Algorithms based on the treewidth of sparse graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 385–396. Springer, Heidelberg (2005)
19. Paulusma, D., van Rooij, J.M.M.: A fast exact algorithm for the 2-role assignment problem (submitted)
20. Randerath, B., Schiermeyer, I.: Exact algorithms for minimum dominating set. Technical Report zaik2004-469, Universität zu Köln, Cologne, Germany (2005)
21. van Rooij, J.M.M., Bodlaender, H.L.: Design by measure and conquer – a faster exact algorithm for dominating set. In: Albers, S., Weil, P. (eds.) Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2008, pp. 657–668. IBFI Schloss Dagstuhl (2008)
22. van Rooij, J.M.M., Bodlaender, H.L., Rossmanith, P.: Dynamic programming on tree decompositions using generalised fast subset convolution. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 566–577. Springer, Heidelberg (2009)
23. van Rooij, J.M.M., Nederlof, J., van Dijk, T.C.: Inclusion/exclusion meets measure and conquer: Exact algorithms for counting dominating set. Technical Report UU-CS-2008-043, Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands (2008)