Dominating Set Counting in Graph Classes*

Shuji Kijima[1], Yoshio Okamoto[2], and Takeaki Uno[3]

[1] Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan kijima@inf.kyushu-u.ac.jp
[2] Center for Graduate Education Initiative, Japan Advanced Institute of Science and Technology, Japan okamotoy@jaist.ac.jp
[3] National Institute of Informatics, Japan uno@nii.ac.jp

Abstract. We make an attempt to understand the dominating set counting problem in graph classes from the viewpoint of polynomial-time computability. We give polynomial-time algorithms to count the number of dominating sets (and minimum dominating sets) in interval graphs and trapezoid graphs. They are based on dynamic programming. With the help of dynamic update on a binary tree, we further reduce the time complexity. On the other hand, we prove that counting the number of dominating sets (and minimum dominating sets) in split graphs and chordal bipartite graphs is #P-complete. These results are in vivid contrast with the recent results on counting the independent sets and the matchings in chordal graphs and chordal bipartite graphs.

1 Introduction

Combinatorics is a branch of mathematics that often deals with counting various objects, and has a long tradition. However, the algorithmic aspect of counting has been less studied. This seems due to the facts that most of the problems turn out to be #P-hard (thus unlikely to have polynomial-time algorithms) and that not many algorithmic techniques have been known.

In the study of graph classes, the situation does not differ. There are many studies on decision problems and optimization problems, but fewer studies on counting problems. Certainly, counting algorithms require properties of graphs that are not needed for solving decision and optimization problems. From this perspective, Okamoto, Uehara and Uno studied two basic counting problems for graph classes. The first paper [8] studied the problem to count the number of independent sets, and provided a linear-time algorithm for chordal graphs. On the other hand, their second paper [7] in the series studied the problems to count the number of matchings and perfect matchings, respectively, and proved that the problem is #P-complete for chordal graphs (actually for split graphs). They are also #P-complete for chordal bipartite graphs. It still remains open whether the number of matchings (or perfect matchings) can be computed in polynomial time for interval graphs, and even for proper interval graphs.

This paper is concerned with dominating sets. In this paper, we will try to understand the dominating set counting problem from the viewpoint of polynomial-time computability. Domination is one of the main subjects in graph theory and graph algorithms,

* The first and second authors are supported by Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science.

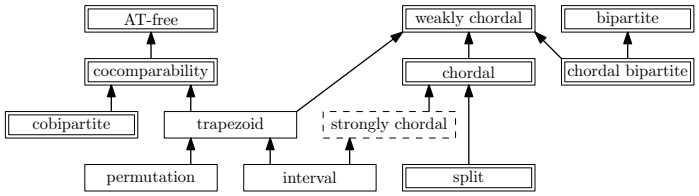


Fig. 1. Complexity landscape for the dominating set counting problem regarding graph classes.

and there are some books especially devoted to that topic [3, 4]. However, not much is known for the counting aspect.

Our result. We give polynomial-time algorithms to count the number of dominating sets (and minimum dominating sets) in interval graphs and trapezoid graphs. They are based on dynamic programming. With the help of dynamic update on a binary tree, we reduce the time complexity. On the other hand, we prove that counting the number of dominating sets in split graphs, chordal bipartite graphs, and cobipartite graphs is #P-complete, and counting the number of minimum dominating sets in split graphs and chordal bipartite graphs is #P-hard.

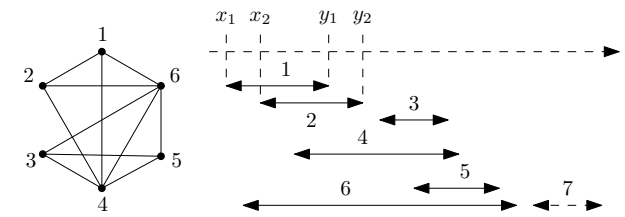
Fig. 1 summarizes our results on counting the number of dominating sets. In the figure, each arrow means that the class on the tail is a subclass of the class on the head. For the graph classes in solid lines polynomial-time algorithms exist, and for the graph classes in double solid lines the problem is #P-complete. For strongly chordal graphs, in a dashed line, the question is left open.

As for counting the minimum dominating sets, Kratsch [5] very recently gave an $\mathrm{O}(n^7)$-time algorithm for AT-free graphs, while the polynomial-time computability is open for strongly chordal graphs. Note that a trapezoid graph is AT-free, but the bound of our algorithm is better than Kratsch's algorithm.

Related work. On counting the number of independent dominating sets and the number of minimum independent dominating sets, Okamoto, Uno and Uehara [8] proved the #P-hardness for chordal graphs, and Lin and Chen [6] gave linear-time algorithms for interval graphs. We note that the parity version of the problem turns out to be trivial: Brouwer, Csorba, and Schrijver [2] proved that every graph has an odd number of dominating sets.

Preliminaries. In this paper, all graphs are finite, simple and undirected. A graph G is denoted by a pair (V, E) of its vertex set V and its edge set E. The set of adjacent vertices of v is called the *neighborhood* of v, and denoted by $N_G(v)$. If there is no confusion, we simply write $N(v)$. For a vertex subset $X \subseteq V$, we denote $N(X) := \left(\bigcup_{v \in X} N(v)\right) \setminus X$.

For $u, v \in V$, we say u *dominates* v if u and v are adjacent in G. A subset $D \subseteq V$ *dominates* v if at least one vertex in D dominates v. A *dominating set* of G is a vertex subset $D \subseteq V$ that dominates all vertices in $V \setminus D$. A dominating set of G is *minimum* if it has the smallest number of elements among all dominating sets of G.



An interval graph G An interval representation of G

Fig. 2. Example of an interval graph and the corresponding intervals.

In the analysis of our algorithms, the time complexity refers to the number of arithmetic operations, not bit operations.

Due to the space constraint, we postpone the proofs of theorems/lemmas with * marks to the full version.

2 Interval graphs

A graph $G = (\{1, \ldots, n\}, E)$ is *interval* if there exists a set $\{I_1, \ldots, I_n\}$ of closed intervals on a real line such that $\{i, j\} \in E$ if and only if $I_i \cap I_j \neq \emptyset$. Such a set of intervals is called an *interval representation* of G. It is known that there always exists an interval representation in which the endpoints of intervals are all distinct, and we can find such a representation of a given interval graph that is sorted by their right endpoints in $\mathrm{O}(n+m)$ time [1], where n is the number of vertices and m is the number of edges. Let $\{I_1, \ldots, I_n\}$ be such a representation, and let each interval be represented as $I_i = [x_i, y_i]$. By the assumption, it holds that $y_i < y_j$ if $i < j$. For the sake of technical simplicity, in the following we add an extra interval $[x_{n+1}, y_{n+1}]$ satisfying that $x_{n+1} > y_i$ for all $i \in \{1, \ldots, n\}$ and $x_{n+1} < y_{n+1}$. See an example in Fig. 2. Note that this addition only doubles the number of dominating sets.

2.1 Counting the dominating sets

The basic strategy is to apply the dynamic programming. To this end, we look at the intervals from left to right, and keep track of the following families of subsets.

Let $F(i, j)$ denote the family of subsets $S \subseteq \{1, \ldots, n\}$ satisfying the following two conditions;

1. $j = \max\{j' \in \{1, \ldots, n\} \mid j' \in S\}$, and
2. $i = \min\{i' \in \{1, \ldots, n+1\} \mid S \text{ does not dominate } i'\}$.

Note that for any $S \subseteq \{1, \ldots, n\}$, there exists a unique pair (i, j) such that $S \in F(i, j)$. Moreover, $S \subseteq \{1, \ldots, n\}$ is a dominating set of the interval graph if and only if $S \in F(n+1, j)$ for some j. Therefore, we readily obtain the following lemma.

Lemma 1. *The number of dominating sets of the interval graph is* $\sum_{j=1}^{n} |F(n+1, j)|$.

□

The next is our key lemma, which indicates an algorithm based on dynamic programming. Let $i^*(j) := \min\{i' \mid i' \notin N(j) \cup \{j\}, i' > j\}$, and let $\mathcal{F} \otimes \{\{j\}\}$ denote $\{S' \cup \{j\} \mid S' \in \mathcal{F}\}$ for $\mathcal{F} \subseteq 2^{\{1,\ldots,j-1\}}$. For instance, in the example of Fig. 2, we have $i^*(1) = 3$ and $i^*(3) = 7$.

Lemma 2. *For $j = 1$,*

- $F(i,1) = \{\{1\}\}$ *if* $i = i^*(1)$, *and*
- $F(i,1) = \emptyset$ *otherwise.*

For $j \in \{2, \ldots, n+1\}$,

- $F(i,j) = \emptyset$ *if* $i \in N(j) \cup \{j\}$,
- $F(i,j) = \left(\bigcup_{j'<j} F(i,j')\right) \otimes \{\{j\}\}$ *if* $i \notin N(j) \cup \{j\}$ *and* $i < j$,
- $F(i,j) = \left(\bigcup_{i' \in N(j) \cup \{j,i\}} \bigcup_{j'<j} F(i',j')\right) \otimes \{\{j\}\}$ *if* $i = i^*(j)$,
- $F(i,j) = \emptyset$ *if* $i > i^*(j)$.

Proof. Consider the case $j = 1$. By the definition of $i^*(1)$, we see that the vertices in $\{1, \ldots, i^*(1) - 1\}$ are dominated by 1. Thus, $F(i^*(1), 1) = \{\{1\}\}$ and $F(i,1) = \emptyset$ when $i \neq i^*(1)$.

Consider next the case $j \geq 2$. We distinguish the following three cases.

Case 1: $i \in N(j) \cup \{j\}$. This means that i is dominated by j, and thus $F(i,j) = \emptyset$.

Case 2: $i \notin N(j) \cup \{j\}$ and $i < j$. For any $S' \in F(i,j')$ with $j' < j$, it holds that $S' \cup \{j\} \in F(i,j)$ since $i \notin N(j) \cup \{j\}$. On the other hand, let $S \in F(i,j)$ be chosen arbitrarily. Since the intervals are sorted by their right endpoints and i is not dominated by S, we see that $i' \notin N(j)$ for any $i' < i$. By the hypothesis on $S \in F(i,j)$, every $i' < i$ must be dominated by $S \setminus \{j\}$. Hence, $S \setminus \{j\} \in F(i,j')$ for some $j' < j$.

Case 3: $i \notin N(j) \cup \{j\}$ and $i > j$. If $i > i^*(j)$, then $F(i,j) = \emptyset$ since $i^*(j)$ is not dominated by j. If $i = i^*(j)$, then by the definition of $i^*(j)$, in a similar way to Case 2, it holds that $F(i,j) = \left(\bigcup_{j'<j} F(i,j') \cup \bigcup_{i' \in N(j) \cup \{j\}} \bigcup_{j'<j} F(i',j')\right) \otimes \{\{j\}\} = \left(\bigcup_{i' \in N(j) \cup \{j,i\}} \bigcup_{j'<j} F(i',j')\right) \otimes \{\{j\}\}$. □

A naïve dynamic-programming algorithm based on the lemma runs in $O(n^3)$ time. In the next section, we discuss how to obtain a faster $O(n \log n)$-time algorithm using a binary-tree data structure.

2.2 Speeding up the dynamic programming

We define $s(i,j) := \sum_{j'=1}^{j} |F(i,j')|$ for any pair of $i, j \in \{1, \ldots, n+1\}$. Then $s(n+1,n) = \sum_{j'=1}^{n} |F(n+1,j')|$ is the number of dominating sets of the interval

graph, from Lemma 1. From Lemma 2, we have $s(i^*(1),1) = |F(i^*(1),1)| = 1$ and $s(i,1) = |F(i,1)| = 0$ for $i \neq i^*(1)$.

Now let $i_{**}(j) := \max\{i' \mid i' \notin N(j) \cup \{j\} \text{ and } i' < j\}$, where we define $i_{**}(j) := 0$ if there is no $i' < j$ such that $i' \notin N(j) \cup \{j\}$. For instance, in the example of Fig. 2, $i_{**}(3) = 2$ and $i_{**}(4) = 0$. Then, any $i \leq i_{**}(j)$ satisfies that $i \notin N(j)$ and $i < j$. Furthermore, any $i \in \{i_{**}(j)+1, \ldots, i^*(j)-1\}$ satisfies that $i \in N(j) \cup \{j\}$. Thus, from Lemma 2, we have the following for $j \in \{2, \ldots, n+1\}$:

- $s(i,j) = 2 \cdot s(i,j-1)$ if $i \leq i_{**}(j)$,
- $s(i,j) = s(i,j-1) + \sum_{i'=i_{**}(j)+1}^{i} s(i',j-1)$ if $i = i^*(j)$,
- $s(i,j) = s(i,j-1)$ otherwise.

This way, we have an $\mathrm{O}(n^2)$-time algorithm to compute $s(n+1,n)$, that is, the number of dominating sets of the interval graph. Namely, we compute $s(1,j), \ldots, s(n+1,j)$ from $s(1,j-1), \ldots, s(n+1,j-1)$ for each $j \in \{2, \ldots, n+1\}$ in $\mathrm{O}(n)$ time. This already improves the bound above.

Now, we discuss how to compute $s(1,j), \ldots, s(n+1,j)$ from $s(1,j-1), \ldots, s(n+1,j-1)$ in $\mathrm{O}(\log n)$ time. To achieve such a running time, we cannot explicitly compute all $s(1,j), \ldots, s(n+1,j)$, since it would take $\Theta(n)$ time. However, the recursive formula above is structured: We either multiply by two over the interval $i \in \{1, \ldots, i_{**}(j)\}$, add the sum of the values $s(i',j-1)$ over the interval $i' \in \{i_{**}(j)+1, \ldots, i^*(j)\}$, or do nothing. This suggests the use of a binary-tree structure to update these values in an implicit way.

We represent the intervals $I_1, \ldots, I_{n+1}$ and the values $s(1,j), \ldots, s(n+1,j)$ in an ordered binary tree T. The tree T has height $\mathrm{O}(\log n)$, with a root r, and with $n+1$ leaves corresponding to the intervals $I_1, \ldots, I_{n+1}$: We name the leaves $1, \ldots, n+1$ and the leaf i corresponds to the interval I_i. The structure of T does not change through computation, but only the values carried by nodes change.

To represent $s(i,j)$, every node $v \in T$ carries two values $w^j(v)$ and $c^j(v)$ satisfying the following three conditions;

$$w^j(v) = c^j(v) \cdot (w^j(u_1) + w^j(u_2)) \quad \text{for every inner node } v, \tag{1}$$

$$s(i,j) = w^j(i) \cdot \prod_{u \in \mathrm{Anc}(i)} c^j(u) \quad \text{for every leaf } i \in \{1, \ldots, n+1\}, \tag{2}$$

where u_1 and u_2 are the children of v, and $\mathrm{Anc}(v)$ denotes the set of proper ancestors of v. When $j = 1$, we set $c^1(v) := 1$ for all $v \in T$, $w^1(v) := 0$ for all $v \neq i^*(1)$, and $w^1(i^*(1)) := 1$. Note that this choice makes the conditions above satisfied when $j = 1$. These values are interpreted as follows. The value $w^j(i)$ plays a role of $s(i,j)$, but since we multiply two in an implicit fashion, we use another value $c^j(i)$ for this purpose, and so $s(i,j)$ cannot be equal to $w^j(i)$, but equal to the product of $w^j(i)$ and the $c^j(u)$ for all proper ancestors u of i.

In our algorithm, we always keep these conditions satisfied, and we only use these conditions for our proof.

Then we observe the following.

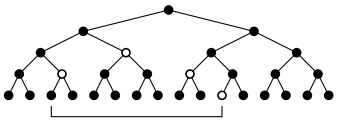


Fig. 3. An example of representing ancestors when $n = 15$. For the leaves $\{3, \ldots, 11\}$, shown by the solid-line interval, the representing ancestors are depicted by white nodes.

Lemma 3 (*). *If w^j and c^j satisfy conditions (1) and (2), then*

$$\sum_{i \in L(v)} s(i,j) = w^j(v) \cdot \prod_{u \in \mathrm{Anc}(v)} c^j(u)$$

holds for any node $v \in T$, where $L(v)$ denotes the set of leaves in the subtree of T rooted at v.

For a consecutive sequence of leaves $\{i_1, \ldots, i_2\}$ of T, we define the *representing ancestors* S_{i_1,i_2} as a minimal subset of nodes in T satisfying that $\bigcup_{v \in S_{i_1,i_2}} L(v) = \{i_1, \ldots, i_2\}$. Note that if $v, v' \in S_{i_1,i_2}$ and $v \neq v'$, then $L(v) \cap L(v') = \emptyset$. Fig. 3 shows an example. Then, we see that

$$\sum_{i=i_1}^{i_2} s(i,j) = \sum_{v \in S_{i_1,i_2}} \sum_{i \in L(v)} s(i,j) = \sum_{v \in S_{i_1,i_2}} w^j(v) \prod_{u \in \mathrm{Anc}(v)} c^j(u),$$

where the last equality follows from Lemma 3.

Note that given a consecutive sequence of leaves $\{i_1, \ldots, i_2\}$ of T, we can find its representing ancestors S_{i_1,i_2} in $\mathrm{O}(\log n)$ time in a bottom-up fashion. This also shows that the number of representing ancestors is $\mathrm{O}(\log n)$.

Now we are ready to describe our algorithm to compute $s(1,j), \ldots, s(n+1,j)$ from $s(1,j-1), \ldots, s(n+1,j-1)$. First, we give a rough outline.

Step 1. We identify $i_{**}(j)$ and $i^*(j)$.
Step 2. For every $i \in \{1, \ldots, i_{**}(j)\}$, let $s(i,j) = 2 \cdot s(i,j-1)$ in an implicit way.
Step 3. We sum up $s(i',j-1)$ over all $i' \in \{i_{**}(j)+1, \ldots, i\}$ in an implicit way, and add it to $s(i,j-1)$ to obtain $s(i,j)$.

For implicit updates, we use the following flush operation. Let i be a leaf of T, and $u_0, \ldots, u_d$ be the sequence of nodes of T from the root r to i satisfying that $u_0 = r$, $u_d = i$ and u_ℓ is a child of $u_{\ell-1}$ for each $\ell \in \{1, \ldots, d\}$. Further, let v_ℓ be the child of $u_{\ell-1}$ different from u_ℓ. The flush operation $\textsc{Flush}(i)$ turns c^j and w^j into $\tilde{c}^j$ and $\tilde{w}^j$ in the following way.

- $\tilde{c}^j(u_\ell) = 1$ for all $\ell \in \{0, \ldots, d-1\}$.
- $\tilde{w}^j(u_\ell) = w^j(u_\ell) \cdot \prod_{u \in \mathrm{Anc}(u_\ell)} c^j(u)$ and $\tilde{w}^j(v_\ell) = w^j(v_\ell) \cdot \prod_{u \in \mathrm{Anc}(v_\ell)} c^j(u)$ for all $\ell \in \{1, \ldots, d\}$.
- $\tilde{c}^j(u) = c^j(u)$ and $\tilde{w}^j(u) = w^j(u)$ for all other u.

Note that the operation can be performed in $\mathrm{O}(\log n)$ time in a top-down manner, since $d = \mathrm{O}(\log n)$. Also observe that $\tilde{c}^j$ and $\tilde{w}^j$ satisfy conditions (1) and (2), and hence $s(i,j) = \tilde{w}^j(i)$.

We now discuss how to perform each step in more detail.

In Step 1, we need to identify $i_{**}(j)$ and $i^*(j)$. To find $i_{**}(j)$, we look for the rightmost interval $\{I_1, \ldots, I_{j-1}\}$ that does not intersect I_j (remind that I_i represents the interval corresponding to the vertex i.) Here, the rightmost interval means the interval with rightmost right endpoint. Note that if I_i intersects I_j and $1 \leq i < i' \leq j-1$, then $I_{i'}$ also intersects I_j. Therefore, $i_{**}(j)$ can be found in $\mathrm{O}(\log n)$ time by binary search.

Similarly, to find $i^*(j)$, we look for the leftmost interval in $\{I_{j+1}, \ldots, I_{n+1}\}$ that does not intersect I_j. If we further store the sorted order of $I_1, \ldots, I_{n+1}$ by their left endpoints, by the same argument as above, we can find $i^*(j)$ in $\mathrm{O}(\log n)$ time.

Before Step 2, we perform $\textsc{Flush}(i_{**}(j))$ and $\textsc{Flush}(i^*(j))$ for later use. This only takes $\mathrm{O}(\log n)$ time. Let $\tilde{c}^{j-1}$ and $\tilde{w}^{j-1}$ be the values obtained by these flush operations. The order of executions of these flushes does not matter: the results are identical, and we only need to ensure that $s(i_{**}(j), j-1) = \tilde{w}^{j-1}(i_{**}(j))$ and $s(i^*(j), j-1) = \tilde{w}^{j-1}(i^*(j))$.

In Step 2, we first identify the representing ancestors $S_{1,i_{**}(j)}$. For later reference, let $S = S_{1,i_{**}(j)}$. Note that at most one representing ancestor can be a leaf of T, in which $i_{**}(j)$ is such a representing ancestor. For every non-leaf representing ancestor $v \in S$, set $c^j(v) = 2 \cdot \tilde{c}^{j-1}(v)$. If $i_{**}(j)$ is a representing ancestor, then set $w^j(i_{**}(j)) = 2 \cdot \tilde{w}^{j-1}(i_{**}(j))$. The whole procedure in Step 2 can be done in $\mathrm{O}(\log n)$ time from the discussion above.

In Step 3, we first identify the representing ancestors $S_{i_{**}(j)+1,i^*(j)-1}$. For later reference, let $S' = S_{i_{**}(j)+1,i^*(j)-1}$. Then, set

$$w^j(i^*(j)) = 2 \cdot \tilde{w}^{j-1}(i^*(j)) + \sum_{v \in S'} \left(\tilde{w}^{j-1}(v) \cdot \prod_{u \in \mathrm{Anc}(v)} \tilde{c}^{j-1}(u) \right).$$

A top-down calculation of the right-hand side only takes $\mathrm{O}(\log n)$ time. Thus, the whole computation in Step 2 can be done in $\mathrm{O}(\log n)$ time.

Finally, if $c^j(v)$ and/or $w^j(v)$ are unset yet, then we keep them unchanged: $c^j(v) = \tilde{c}^{j-1}(v)$ and $w^j(v) = \tilde{w}^{j-1}(v)$, respectively. Then, we adjust the values $w^j(u)$ for all $u \in S \cup \bigcup_{v \in S} \mathrm{Anc}(v) \cup \mathrm{Anc}(i^*(j))$ in a bottom-up manner to keep condition (1) satisfied. This can also be done in $\mathrm{O}(\log n)$ time: For unset values, we do not have to set the values explicitly, but we just keep these values; For the adjustment, observe that the number of nodes in $S \cup \bigcup_{v \in S} \mathrm{Anc}(v) \cup \mathrm{Anc}(i^*(j))$ is $\mathrm{O}(\log n)$. Note that $w^j(u) = \tilde{w}^{j-1}(u)$ for every node u that does not belong to $S \cup \bigcup_{v \in S} \mathrm{Anc}(v) \cup \mathrm{Anc}(i^*(j))$.

This completes the description of the whole algorithm. The running time amounts to $\mathrm{O}(\log n)$, and after iterating over $j \in \{2, \ldots, n\}$, we get $\mathrm{O}(n \log n)$ as the overall time complexity.

We now prove its correctness. Note that after the whole procedure condition (1) is satisfied. Therefore, it suffices to prove that condition (2) is satisfied. If $i < i_{**}(j)$, then

there exists a unique ancestor $v \in \mathrm{Anc}(i)$ such that $v \in S$ and $i \in L(v)$. Hence,

$$\begin{aligned} s(i,j) &= 2 \cdot s(i,j-1) = 2 \cdot \tilde{w}^{j-1}(i) \cdot \prod_{u \in \mathrm{Anc}(i)} \tilde{c}^{j-1}(u) \\ &= \tilde{w}^{j-1}(i) \cdot (2 \cdot \tilde{c}^{j-1}(v)) \cdot \prod_{u \in \mathrm{Anc}(i) \setminus \{v\}} \tilde{c}^{j-1}(u) \\ &= w^j(i) \cdot c^j(v) \cdot \prod_{u \in \mathrm{Anc}(i) \setminus \{v\}} c^j(u) = w^j(i) \cdot \prod_{u \in \mathrm{Anc}(i)} c^j(u). \end{aligned}$$

If $i = i_{**}(j)$, then we have two cases. If $i_{**}(j) \notin S$, then we can proceed as above for $i < i_{**}(j)$. If $i_{**}(j) \in S$, then $s(i,j) = 2 \cdot s(i,j-1) = 2 \cdot \tilde{w}^{j-1}(i) = w^j(i) = w^j(i) \cdot \prod_{u \in \mathrm{Anc}(i)} c^j(u)$, where the second equality holds since $\textsc{Flush}(i_{**}(j))$ was performed, and the last equality holds since $c^j(u) = \tilde{c}^{j-1}(u) = 1$ due to $\textsc{Flush}(i_{**}(j))$.

If $i = i^*(j)$, then

$$\begin{aligned} s(i^*(j),j) &= s(i^*(j),j-1) + \sum_{i'=i_{**}(j)+1}^{i^*(j)} s(i',j-1) \\ &= 2 \cdot s(i^*(j),j-1) + \sum_{i'=i_{**}(j)+1}^{i^*(j)-1} s(i',j-1) \\ &= 2 \cdot \tilde{w}^{j-1}(i^*(j)) + \sum_{v \in S'} \tilde{w}^{j-1}(v) \prod_{u \in \mathrm{Anc}(v)} \tilde{c}^{j-1}(u) \\ &= 2 \cdot \tilde{w}^{j-1}(i^*(j)) + (w^j(i^*(j)) - 2 \cdot \tilde{w}^{j-1}(i^*(j))) \\ &= w^j(i^*(j)) = w^j(i^*(j)) \cdot \prod_{u \in \mathrm{Anc}(i^*(j))} c^j(u), \end{aligned}$$

where the last equality holds since $c^j(u) = \tilde{c}^{j-1}(u) = 1$ due to $\textsc{Flush}(i^*(j))$.

Thus, we conclude that our algorithm correctly computes c^j and w^j that satisfy conditions (1) and (2). After iterating the procedure in the increasing order of $j \in \{2, \ldots, n\}$, we output $s(n+1, n)$. This is the number of dominating sets. Thus, we obtain the following theorem.

Theorem 1. *We can count the number of dominating sets of an n-vertex interval graph in $\mathrm{O}(n \log n)$ time, when the graph is given by a set of n intervals sorted by their right endpoints.* □

2.3 Counting the minimum dominating sets

By slightly modifying the recursion formulas in Lemma 2, we obtain a polynomial-time algorithm for counting the number of dominating sets of each possible size in an interval graph. We define $F'(i,j,k)$ as $\{S \in F(i,j) \mid |S| = k\}$. Then we have the following.

Lemma 4. *For $j = 1$,*

– $F'(i,1,1) = \{\{1\}\}$ *if* $i = i^*(1)$, *and*
– $F'(i,1,k) = \emptyset$ *otherwise.*

For $j \in \{2,\ldots,n+1\}$,

– $F'(i,j,k) = \emptyset$ *if* $i \in N(j) \cup \{j\}$;
– $F'(i,j,k) = \left(\bigcup_{j'<j} f'(i,j',k-1)\right) \otimes \{\{j\}\}$ *if* $i \notin N(j) \cup \{j\}$ *and* $i < j$;
– $F'(i,j,k) = \left(\bigcup_{i' \in N(j) \cup \{j,i\}} \bigcup_{j'<j} f'(i',j',k-1)\right) \otimes \{\{j\}\}$ *if* $i = i^*(j)$;
– $F'(i,j,s) = \emptyset$ *if* $i > i^*(j)$. □

Based on this lemma, we obtain the following theorem in a similar way to the dominating set counting problem.

Theorem 2. *Given an n-vertex interval graph, we can count the number of dominating sets of each possible size, hence the number of minimum dominating sets, in $\mathrm{O}(n^2 \log n)$ time.* □

In a similar way, we can deal with trapezoid graphs.

Theorem 3 (*). *Given an n-vertex trapezoid graph, we can count the number of dominating sets in $\mathrm{O}(n^3 \log n)$ time, and the number of dominating sets of each possible size, hence the number of minimum dominating sets, in $\mathrm{O}(n^4 \log n)$ time.*

3 Split graphs

A graph is *split* if its vertex set can be partitioned into a clique and an independent set. In other words, it can be obtained from a bipartite graph by adding edges to turn one partite set into a clique. Note that a split graph is always chordal.

Theorem 4. *Counting the number of dominating sets in a split graph is #P-complete.*

Proof. We reduce the following problem to the problem in the statement: Count the number of independent sets in a bipartite graph. This is known to be #P-complete [9]. Recall that an *independent set* in a graph is a vertex subset that induces no edge.

Let $G = (U, V; E)$ be a given bipartite graph where $|U| = n$ and $|V| = m$. Without loss of generality, we assume that G has no isolated vertex, and thus $N_G(V) = U$. (Otherwise, let k be the number of isolated vertices in G. Then, the number of independent sets is 2^k times the number of independent sets in the graph obtained from G by deleting all isolated vertices.)

Consider an independent set S of G, and let $X := S \cap U$. Then, $(S \cap V) \cap N_G(X) = \emptyset$. Namely, $S \cap V \subseteq V \setminus N_G(X)$. Therefore, S gives rise to a pair $(X, Y) \in 2^U \times 2^V$ such that $Y = S \cap V \subseteq V \setminus N_G(X)$. On the other hand, for any pair $(X, Y) \in 2^U \times 2^V$ such that $Y \subseteq V \setminus N_G(X)$, we can see that $X \cup Y$ is independent in G. This means that $S \leftrightarrow (X, Y)$ is a one-to-one correspondence, and this correspondence yields the following formula if we denote by $\mathsf{ind}(G)$ the number of independent sets in

G: $\mathsf{ind}(G) = |\{(X,Y) \in 2^U \times 2^V \mid Y \subseteq V \setminus N_G(X)\}| = \sum_{X \subseteq U} 2^{|V \setminus N_G(X)|}$. Let $\nu_i := |\{X \subseteq U \mid N_G(X) = i\}|$. Then, the above formula can be rephrased as $\mathsf{ind}(G) = 2^m \sum_{i=0}^{m} \nu_i 2^{-i}$. Therefore, if we are able to know $\nu_0, \nu_1, \ldots, \nu_m$ in polynomial time, then we can retrieve $\mathsf{ind}(G)$ in polynomial time.

Let G_1 be a split graph constructed from G by filling the edges between all pairs of vertices in U.

Let D be a dominating set of G_1. Then, there exists a unique pair of sets $(X,Y) \in 2^U \times 2^V$ such that $D = X \dot\cup Y \dot\cup (V \setminus N_G(X))$. Note that this is also valid when $X = \emptyset$ since we have assumed that $N_G(V) = U$. Also note that $Y \subseteq N_G(X)$ (reminder: "$\dot\cup$" means a disjoint union). On the other hand, for any $X \subseteq U$ and any $Y \subseteq N_G(X)$, the set $D = X \dot\cup Y \dot\cup (V \setminus N_G(X))$ is a dominating set of G. Therefore, $D \leftrightarrow (X,Y)$ is a one-to-one correspondence, and this correspondence yields the following formula if we denote by $\mathsf{dom}(G_1)$ the number of independent sets in G_1: $\mathsf{dom}(G_1) = \sum_{X \subseteq U} 2^{|N_G(X)|} = \sum_{i=0}^{m} \nu_i 2^i$.

More generally, we construct a split graph G_j from G as follows. The vertex set of G_j consists of U and j copies $V_1, \ldots, V_j$ of V. We denote the copy of $v \in V$ in $V_{j'}$ by $v_{j'}$. As for the edges, every pair of vertices in U is joined by an edge. A vertex $u \in U$ is joined by an edge with $v_{j'} \in V_{j'}$ if and only if $\{u,v\}$ is an edge of G.

Let D be a dominating set of G_j. Then, there exists a $(j+1)$-tuple $(X, Y_1, \ldots, Y_j) \in 2^U \times 2^{V_1} \times \cdots \times 2^{V_j}$ such that $D = X \dot\cup \dot\bigcup_{j'} (Y_{j'} \dot\cup (V_{j'} \setminus N_G(X)))$. We can see this is a one-to-one correspondence as in the argument for G_1. Therefore, we obtain $\mathsf{dom}(G_j) = \sum_{X \subseteq U} \left(2^{|N_G(X)|}\right)^j = \sum_{i=0}^{m} \nu_i (2^j)^i$.

Thus, $\mathsf{dom}(G_j)$ is represented as a linear combination of $\nu_0, \ldots, \nu_m$ for all $j = 1, \ldots, m+1$. We want to extract $\nu_0, \ldots, \nu_m$ from these combinations. This can be done if we regard them as a system of linear equations in which $\nu_0, \ldots, \nu_m$ are variables. It turns out that the coefficient matrix in the system is a Vandermonde matrix, and thus non-singular. Therefore, if we have $\mathsf{dom}(G_j)$ for all $j \in \{1, \ldots, m+1\}$ in polynomial time, then we obtain ν_i for all $i \in \{0, \ldots, m\}$ in polynomial time. This concludes the reduction. (This proof technique is often called the *interpolation method*.) □

Theorem 5 (*). *Counting the number of minimum dominating sets in a split graph is #P-hard.*

4 Cobipartite graphs

A graph is *cobipartite* if it is the complement of a bipartite graph. Namely, the vertex set can be partitioned into two cliques. Note that a cobipartite graph is cocomparability graph, and thus AT-free.

Theorem 6 (*). *Counting the number of dominating sets in a cobipartite graph is #P-complete.*

Note that we can easily count the number of minimum dominating sets in a cobipartite graph, since the size of a minimum dominating set is always at most two.

5 Chordal bipartite graphs

A graph is *chordal bipartite* if it is bipartite and every induced cycle is of length four.

Theorem 7. *Counting the number of dominating sets in a chordal bipartite graph is #P-complete.*

Proof. We reduce the following problem to the problem in the statement: Count the number of independent sets in a bipartite graph. This is known to be #P-complete [9].

Let $G = (U, V; E)$ be a given bipartite graph. We construct a chordal bipartite graph $G_{i,j}$ as follows. The vertex set of $G_{i,j}$ is $U \cup V \cup \{a_{u,j'} \mid u \in U, j' \in \{1, \dots, j\}\} \cup \{b_{v,j'} \mid v \in V, j' \in \{1, \dots, j\}\} \cup \{p^e_{i'} \mid e \in E, i' \in \{1, \dots, i\}\} \cup \{q^e_{i'} \mid e \in E, i' \in \{1, \dots, i\}\}$. The edge set of $G_{i,j}$ is $\{\{u, v\} \mid u \in U, v \in V\} \cup \{\{a_{u,j'}, u\} \mid u \in U, j' \in \{1, \dots, j\}\} \cup \{\{b_{v,j'}, v\} \mid v \in V, j' \in \{1, \dots, j\}\} \cup \{\{p^e_{i'}, q^e_{i'}\} \mid e \in E, i' \in \{1, \dots, i\}\} \cup \{\{u, p^e_{i'}\} \mid u \in U, e \in E, u \in e, i' \in \{1, \dots, i\}\} \cup \{\{v, q^e_{i'}\} \mid v \in V, e \in E, v \in e, i' \in \{1, \dots, i\}\}$.

Let D be a dominating set of $G_{i,j}$. Let $X = D \cap U$ and $Y = D \cap V$. Then, it must hold that $a_{u,j'} \in D$ for all $u \in U \setminus X, j' \in \{1, \dots, j\}$, and $b_{v,j'} \in D$ for all $v \in V \setminus Y, j' \in \{1, \dots, j\}$. For an edge $e = \{u, v\} \in E$ such that $u \notin X$ or $v \notin Y$, at least one of $p^e_{i'}$ and $q^e_{i'}$ should belong to D for every $i' \in \{1, \dots, i\}$. Therefore, for fixed $X \subseteq U$ and $Y \subseteq V$, the number of dominating sets D such that $X = D \cap U$ and $Y = D \cap V$ is equal to

$$2^{j|X|} \cdot 2^{j|Y|} \cdot \prod_{\substack{\{u,v\} \in E, \\ u \in X, v \in Y}} 2^{2i} \prod_{\substack{\{u,v\} \in E, \\ u \notin X \text{ or } v \notin Y}} 3^i = 2^{j(|X|+|Y|)} \cdot 3^{i|E|} \cdot \left(\frac{4}{3}\right)^{i \cdot e(X,Y)},$$

where $e(X, Y) = |\{\{u, v\} \in E \mid u \in X, v \in Y\}|$. Therefore, the number of dominating sets in $G_{i,j}$ is equal to

$$\begin{aligned} \mathsf{dom}(G_{i,j}) &= \sum_{X \subseteq U} \sum_{Y \subseteq V} 2^{j(|X|+|Y|)} \cdot 3^{i|E|} \cdot \left(\frac{4}{3}\right)^{i \cdot e(X,Y)} \\ &= \sum_{k=0}^{|E|} \underbrace{\left(\sum_{\substack{X \subseteq U, Y \subseteq V, \\ e(X,Y) = k}} 2^{j(|X|+|Y|)} \right)}_{=: \delta_{j,k}} \left(3^{|E|} \left(\frac{4}{3}\right)^k \right)^i. \end{aligned}$$

Then, from the interpolation method, by solving this system of linear equations, we can retrieve $\delta_{j,0}, \dots, \delta_{j,|E|}$ from $\mathsf{dom}(G_{1,j}), \dots, \mathsf{dom}(G_{|E|+1,j})$ in polynomial time.

Now notice that

$$\delta_{j,0} = \sum_{\substack{X \subseteq U, Y \subseteq V, \\ e(X,Y) = 0}} 2^{j(|X|+|Y|)} = \sum_{\ell=0}^{|U|+|V|} 2^{j\ell} \alpha_\ell,$$

where α_ℓ denotes the number of independent sets of size ℓ in G, since $X \cup Y$ is independent in G if and only if $e(X, Y) = 0$. Then, by the interpolation method, we can retrieve $\alpha_0, \ldots, \alpha_{|U|+|V|}$ from $\delta_{1,0}, \ldots, \delta_{|U|+|V|+1,0}$ in polynomial time. Thus, the reduction is completed. □

Theorem 8 (*). *Counting the number of minimum dominating sets in a chordal bipartite graph is #P-hard.*

6 Concluding Remarks

We have investigated the dominating set counting problem in graph classes from the viewpoint of polynomial-time computability. An obvious open question is to determine the complexity of the dominating set counting problem and the minimum dominating set counting problem for strongly chordal graphs. Further investigation should be done from the viewpoints of approximation and fixed-parameter tractability, too.

A similar question can be asked for counting the number of minimal dominating sets. In the full version, we prove #P-hardness of the problem in cobipartite graphs and chordal bipartite graphs.

References

1. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. Journal of Computer and System Sciences **13** (1976) 335-379.
2. A. E. Brouwer, P. Csorba, and A. Schrijver. The number of dominating sets of a finite graph is odd. Preprint, 2009. `http://www.win.tue.nl/~aeb/preprints/domin4a.pdf`
3. T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, eds. *Fundamentals of Domination in Graphs*. Marcel Dekker, 1998.
4. T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, eds. *Domination in Graphs: Advanced Topics*. Marcel Dekker, 1998.
5. D. Kratsch. Personal communication, April 2011.
6. M.-S. Lin and Y.-J. Chen. Linear time algorithms for counting the number of minimal vertex covers with minimum/maximum size in an interval graph. Information Processing Letters **107** (2008) 257–264.
7. Y. Okamoto, R. Uehara, and T. Uno. Counting the number of matchings in chordal and chordal bipartite graph classes. Proc. WG 2009, 296–307.
8. Y. Okamoto, T. Uno, and R. Uehara. Counting the number of independent sets in chordal graphs. Journal of Discrete Algorithms **6** (2008) 229–242.
9. J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. SIAM Journal on Computing **12** (1983) 777–788.