LCNS, Vol 2286, pp. 613–627, Springer 2002

Improved Tree Decomposition Based Algorithms for Domination-like Problems

Jochen Alber* and Rolf Niedermeier

Wilhelm-Schickard-Institut für Informatik,
Universität Tübingen,
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany,
email: {alber,niedermr}@informatik.uni-tuebingen.de

Abstract. We present an improved dynamic programming strategy for DOMINATING SET and related problems on graphs that are given together with a tree decomposition of width k. We obtain an $O(4^k n)$ algorithm for DOMINATING SET, where n is the number of nodes of the tree decomposition. This result improves the previously best known algorithm of Telle and Proskurowski running in time $O(9^k n)$. The key to our result is an argument on a certain "monotonicity" in the table updating process during dynamic programming.
Moreover, various other domination-like problems as discussed by Telle and Proskurowski are treated with our technique. We gain improvements on the base of the exponential term in the running time ranging between 55% and 68% in most of these cases. These results mean significant breakthroughs concerning practical implementations.

Classification: algorithms and data structures, combinatorics and graph theory, computational complexity.

1 Introduction

Solving domination-like problems on graphs developed into a research field of its own [12, 13]. According to a year 1998 survey [12, Chapter 12], more than 200 research papers have been published on the algorithmic complexity of domination and related graph problems. Since these problems turn out to be very hard and are even NP-complete for most special graph classes (cf. [8, 16]), a main road of attack against their intractability has been through studying their complexity for graphs of bounded treewidth (or, equivalently, partial k-trees), see, e.g., [1, 2, 10, 20, 21]. It is well-known that for graphs of bounded treewidth k, DOMINATING SET can be solved in time exponential in k, but linear in the size of the tree decomposition [7]. Hence, a central question is to get the combinatorial explosion in k as small as possible—this is what we investigate here, significantly improving previous work [10, 20, 21].

* Supported by the Deutsche Forschungsgemeinschaft (DFG), research project PEAL (Parameterized complexity and Exact ALgorithms), NI 369/1-1.

Before describing previous work and our results in more detail, let us briefly define the core problem Dominating Set. An *h-dominating set* D of an undirected graph G is a set of h vertices of G such that each of the rest of the vertices has at least one neighbor in D. The minimum h such that the graph G has an h-dominating set is called the *domination number* of G, denoted by $\gamma(G)$. The dominating set problem is to decide, given a graph $G = (V, E)$ and a positive integer h, whether or not there exists an h-dominating set. In the course of this paper, firstly, we will concentrate on Dominating Set and, secondly, we show how to extend our findings to several other domination-like problems. To better understand our results, a few words about tree decomposition (formally defined in Section 2) based algorithms seem appropriate. Typically, treewidth based algorithms proceed according to the following scheme in two stages:

1. Find a tree decomposition of bounded width for the input graph, and then
2. solve the problem using dynamic programming approaches on the tree decomposition (see [7]).

As to the first stage, when given a graph G and an integer k, the problem to determine whether the treewidth of G is at most k, is NP-complete. When the parameter k is a fixed constant, however, a lot of work was done on polynomial time solutions, culminating in Bodlaender's celebrated linear time algorithm [6]—with a hidden constant factor exponential in k that still seems too large for practical purposes. That is why also heuristic approaches for constructing tree decompositions are in use, see [15] for an up-to-date account. In our own recent research on fixed parameter algorithms for problems on planar graphs [1, 2] together with corresponding implementation work [3] (based on LEDA [17]) we are currently doing, it turned out that, the first stage above usually can be done very quickly, the computational bottleneck being the second stage. More precisely, in [2] we showed that planar graph input instances $G = (V, E)$ of a parameterized problem that satisfies the so-called Layerwise Separation Property (these problems include, e.g., dominating set, vertex cover, independent set) allow for tree decompositions of width $k = O(\sqrt{h})$, where h is the problem parameter (such as the size of the dominating set, vertex cover, independent set, etc.). This tree decomposition can be constructed in time $O(\sqrt{h}|V|)$. Our work in progress concerning experimental studies on random planar graph instances shows that, in this context, for parameter values $h \approx 1000$ (independent of the graph size), we are confronted with tree decompositions of width approximately 15 to 20 [3]. This stirred our interest in research on "dynamic programming on tree decompositions," the results of which we report upon in the following.

To our knowledge, the best previous results for (dynamic programming) algorithms on tree decompositions applied to domination-like problems were obtained by Telle and Proskurowski [20, 21]. For the time being, let us concentrate on Dominating Set itself. For a graph with a tree decomposition of n tree nodes and width k, Telle and Proskurowski solve Dominating Set in time $O(9^k n)$. Ten years earlier, Corneil and Keil [10] presented an $O(4^k n^{k+2})$ algorithm for k-trees. Observe, however, that the latter algorithm is *not* "fixed parameter tractable" in the sense of parameterized complexity theory [11], since

Algorithm	$k = 5$	$k = 10$	$k = 15$	$k = 20$
$9^k n$	0.05 sec	1 hour	6.5 years	$3.9 \cdot 10^5$ years
$4^k n$	0.001 sec	1 sec	18 minutes	13 days

Table 1. Comparing our $O(4^k n)$ algorithm for DOMINATING SET with the $O(9^k n)$ algorithm of Telle and Proskurowski in the case $n = 1000$ (number of nodes of the tree decomposition), we assume a machine executing 10^9 instructions per second and we neglect the constants hidden in the O-terms (which are comparable in both cases).

its running time is not of the form "$f(k)n^{O(1)}$," where f may be an arbitrary function depending only on k. Our new result is to improve the running time from $O(9^k n)$ to $O(4^k n)$, a significant reduction of the combinatorial explosion.[1] We achieve this by introducing a new, in a sense general concept of "monotonicity" for dynamic programming for domination-like problems. Using this concept, we can improve basically all running times for domination-like problems given by Telle and Proskurowski (see Table 2 in the concluding section for a complete overview). For example, we can solve the so-called TOTAL DOMINATING SET problem in time $O(5^k n)$, where Telle and Proskurowski had $O(16^k n)$.

To illustrate the significance of our results, in Table 1, we compare (hypothetical) running times of the $O(9^k n)$ algorithm of Telle and Proskurowski [20, 21] to our $O(4^k n)$ algorithm for some realistic values of k and $n = 1000$; we assume a fixed underlying machine with 10^9 instructions per second. It is worth noting that improving exponential terms is a "big issue" for fixed parameter algorithms. For example, much attention was paid to lowering the exponential term for VERTEX COVER (for general graphs) from a trivial 2^h to below 1.3^h [9, 18], where h is the size of a minimum vertex cover. By way of contrast, here we obtain much more drastic improvements, which, additionally, even apply to a whole class of problems.

2 Preliminaries

The main tool we use in our algorithm is the concept of tree decompositions as, e.g., described in [7, 14].

Definition 2.1. Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\langle \{X_i \mid i \in I\}, T \rangle$, where each X_i is a subset of V, called a *bag*, and T is a tree with the elements of I as nodes. The following three properties must hold:

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

The *width* of $\langle \{X_i \mid i \in I\}, T \rangle$ equals $\max\{|X_i| \mid i \in I\} - 1$. The *treewidth* of G is the minimum k such that G has a tree decomposition of width k.

[1] Observe that in our work [1] we gave wrong bounds for the dynamic programming algorithm for DOMINATING SET, claiming a running time $O(3^k n)$. This was due to a misinterpretation of [21, Theorem 5.7] where the correct base of the exponential term in the running time is $3^2 = 9$ instead of 3.

A tree decomposition with a particularly simple structure is given by the following.

Definition 2.2. A tree decomposition $\langle \{X_i \mid i \in I\}, T \rangle$ is called a *nice tree decomposition* if the following conditions are satisfied:

1. Every node of the tree T has at most 2 children.
2. If a node i has two children j and k, then $X_i = X_j = X_k$ (in this case i is called a JOIN NODE).
3. If a node i has one child j, then one of the following situations must hold
   (a) $|X_i| = |X_j| + 1$ and $X_j \subset X_i$ (in this case i is called an INTRODUCE NODE), or
   (b) $|X_i| = |X_j| - 1$ and $X_i \subset X_j$ (in this case i is called a FORGET NODE).

It is not hard to transform a given tree decomposition into a nice tree decomposition. More precisely, the following result holds (see [14, Lemma 13.1.3]).

Lemma 2.3. *Given a width k and n nodes tree decomposition of a graph G, one can find a width k and $O(n)$ nodes nice tree decomposition of G in linear time.* □

3 Dynamic Programming Based on "Monotonicity"

In this section, we present our main algorithm which is based on a fresh view on dynamic programming. Compared to previous work, we perform the updating process of the tables in a more careful, less time consuming way by making use of the "monotonous structure of the tables." Our main result is as follows.

Theorem 3.1. *If a width k tree decomposition of a graph is known, then a minimum dominating set can be determined in time $O(4^k n)$, where n is the number of nodes of the tree decomposition.*

The outline of the corresponding algorithm and its proof of correctness fill the rest of this section. From now on suppose that the given tree decomposition of our input graph $G = (V, E)$ is $\mathcal{X} = \langle \{X_i \mid i \in I\}, T \rangle$. By Lemma 2.3, we can assume that $\mathcal{X}$ is a nice tree decomposition.

3.1 Colorings and Monotonicity

Suppose that $V = \{x_1, \ldots, x_n\}$. We assume that the vertices in the bags are ordered by their indices, i.e., $X_i = (x_{i_1}, \ldots, x_{i_{n_i}})$ with $i_1 \leq \ldots \leq i_{n_i}$ for all $i \in I$.

Colorings. In the following, we use three different "colors" that will be assigned to the vertices in a bag:

- "black" (represented by 1, meaning that the vertex belongs to the dominating set),
- "white" (represented by 0, meaning that the vertex is already dominated at the current stage of the algorithm), and
- "grey" (represented by $\hat{0}$, meaning that, at the current stage of the algorithm, we still ask for a domination of this vertex).

A vector $c = (c_1, \ldots, c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$ will be called a *coloring* for the bag $X_i = (x_{i_1}, \ldots, x_{i_{n_i}})$, and the *color* assigned to vertex x_{i_t} by the coloring c is given by the coordinate c_t.

For each bag X_i (with $|X_i| = n_i$), we will use a mapping

$$A_i : \{0, \hat{0}, 1\}^{n_i} \longrightarrow \mathbb{N} \cup \{+\infty\}.$$

For a coloring $c = (c_1, \ldots, c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$, the value $A_i(c)$ stores how many vertices are needed for a minimum dominating set (of the graph visited up to the current stage of the algorithm) under the restriction that the color assigned to vertex x_{i_t} is c_t $(t = 1, \ldots, n_i)$.

A coloring $c \in \{0, \hat{0}, 1\}^{n_i}$ is *locally invalid* for a bag X_i if

$$\big(\exists s \in \{1, \ldots, n_i\} : c_s = 0\big) \wedge \big(\nexists t \in \{1, \ldots, n_i\} : (x_{i_t} \in N(x_{i_s}) \wedge c_t = 1)\big).$$

In other words, a coloring is locally invalid if there is some vertex x_{i_s} in the bag that is colored white, but this color is not "justified" within the bag, i.e., x_{i_s} is not dominated by a vertex within the bag using this coloring.[2] Also, for a coloring $c = (c_1, \ldots, c_m) \in \{0, \hat{0}, 1\}^m$ and a color $d \in \{0, \hat{0}, 1\}$, we use the notation

$$\#_d(c) := |\{t \in \{1, \ldots, m\} : c_t = d\}|.$$

Monotonicity. On the color set $\{0, \hat{0}, 1\}$, let $\prec$ be the partial ordering given by $\hat{0} \prec 0$ and $d \prec d$ for all $d \in \{0, \hat{0}, 1\}$. This ordering naturally extends to colorings: For $c = (c_1, \ldots, c_m), c' = (c'_1, \ldots, c'_m) \in \{0, \hat{0}, 1\}^m$, we let $c \prec c'$ iff $c_t \prec c'_t$ for all $t = 1, \ldots, m$.

We call a mapping

$$A_i : \{0, \hat{0}, 1\}^{n_i} \longrightarrow \mathbb{N} \cup \{+\infty\}$$

monotonous from the partially ordered set $(\{0, \hat{0}, 1\}^{n_i}, \prec)$ to $(\mathbb{N} \cup \{+\infty\}, \leq)$ if for $c, c' \in \{0, \hat{0}, 1\}^{n_i}$, $c \prec c'$ implies $A(c) \leq A(c')$. It is very essential for the correctness of our algorithm as well as for the claimed running time that all the mappings A_i will be monotonous.

[2] A locally invalid coloring still may be a correct coloring if the white vertex whose color is not "justified" *within* the bag already is dominated by a vertex from other bags.

3.2 The Algorithm

We use the mappings introduced above to perform a dynamic programming approach. Note that at each stage of the algorithm the mappings visited up to that stage are monotonous. This is guaranteed by Lemmas 3.2, 3.3, 3.4, and 3.5.

Step 1 (initialization). In the first step of the algorithm, for each leaf node i of the tree decomposition, we initialize the mapping A_i:

$$\texttt{for all } c \in \{0, \hat{0}, 1\}^{n_i} \texttt{ do}$$
$$A_i(c) \leftarrow \begin{cases} +\infty & \text{if } c \text{ is locally invalid for } X_i \\ \#_1(c) & \text{otherwise} \end{cases} \tag{1}$$

By this initialization step, we make sure that only colorings are taken into consideration where an assignment of color 0 is justified.

It is trivial to observe the following.

Lemma 3.2. *1. The evaluations in (1) can be carried out in time $O(3^{n_i} n_i)$.*
2. The mapping A_i is monotonous. □

Step 2 (updating process). After the initialization, we visit the bags of our tree decomposition bottom-up from the leaves to the root, evaluating the corresponding mappings in each step according to the following rules.

FORGET NODES: Suppose i is a FORGET NODE with child j and suppose that $X_i = (x_{i_1}, \ldots, x_{i_{n_i}})$. W.l.o.g.[3], we may assume that $X_j = (x_{i_1}, \ldots, x_{i_{n_i}}, x)$. Evaluate the mapping A_i of X_i as follows:

$$\texttt{for all } c \in \{0, \hat{0}, 1\}^{n_i} \texttt{ do}$$
$$A_i(c) \leftarrow \min_{d \in \{0,1\}} A_j(c \times \{d\}) \tag{2}$$

Note that a coloring $c \times \{\hat{0}\}$ for X_j means that the vertex x is assigned color $\hat{0}$, i.e., not yet dominated by a vertex. Since, by condition 3. of Definition 2.1, the vertex x will never appear in a bag for the rest of the algorithm, a coloring $c \times \{\hat{0}\}$ will remain unresolved and it will not lead to a dominating set. That is why the minimum in the assignment (2) is taken over colors 1 and 0 only.

The following lemma is easy to see.

Lemma 3.3. *1. The evaluations in (2) can be carried out in time $O(3^{n_i})$.*
2. If the mapping A_j is monotonous, then mapping A_i also is monotonous. □

INTRODUCE NODES: Suppose that i is an INTRODUCE NODE with child j and suppose, furthermore, that $X_j = (x_{j_1}, \ldots, x_{j_{n_j}})$. W.l.o.g.[4], we may assume

[3] Possibly after rearranging the vertices in X_j and the entries of A_j accordingly.
[4] Possibly after rearranging the vertices in X_i and the entries of A_i accordingly.

that $X_i = (x_{j_1}, \ldots, x_{j_{n_j}}, x)$. Let $N(x) \cap X_i = \{x_{j_{p_1}}, \ldots, x_{j_{p_s}}\}$ be the neighbors of the "introduced" vertex x that appear in the bag X_i. We now define a function $\phi : \{0, \hat{0}, 1\}^{n_j} \to \{0, \hat{0}, 1\}^{n_j}$ on the set of colorings of X_j. For $c = (c_1, \ldots, c_{n_j}) \in \{0, \hat{0}, 1\}^{n_j}$, let $\phi(c) := (c'_1, \ldots, c'_{n_j})$ such that

$$c'_t = \begin{cases} \hat{0} \text{ if } t \in \{p_1, \ldots, p_s\} \text{ and } c_t = 0, \\ c_t \text{ otherwise.} \end{cases}$$

Then, evaluate the mapping A_i of X_i as follows:

`for all` $c = (c_1, \ldots, c_{n_j}) \in \{0, \hat{0}, 1\}^{n_j}$ `do`

$$A_i(c \times \{0\}) \leftarrow \begin{cases} A_j(c) \text{ if } x \text{ has a neighbor } x_{j_q} \text{ in } X_i \text{ with } c_q = 1, \\ +\infty \quad \text{otherwise} \end{cases} \tag{3}$$

$$A_i(c \times \{1\}) \leftarrow A_j(\phi(c)) + 1 \tag{4}$$

$$A_i(c \times \{\hat{0}\}) \leftarrow A_j(c) \tag{5}$$

For the correctness of the assignments (3) and (4), we remark the following: It is clear that, if we assign color 0 to vertex x (see assignment (3)), we again (as already done in the initializing step of assignment (1)) have to check whether this color can be justified at the current stage of the algorithm. Such a justification is given if and only if the coloring under examination already assigns a 1 to some neighbor of x in X_i. This is true, since condition 3. of Definition 2.1 implies that no neighbor of x has been considered in previous bags, and, hence, up to the current stage of the algorithm, x can only be dominated by a vertex in X_i (as checked in assignment (3)).
If we assign color 1 to vertex x (see assignment (4)), we already dominate all vertices $\{x_{j_{p_1}}, \ldots, x_{j_{p_s}}\}$. Suppose now we want to evaluate $A_i(c \times \{1\})$ and suppose some of these vertices are assigned color 0 by c, say $c_{p'_1} = \ldots = c_{p'_q} = 0$ (where $(p'_1, \ldots, p'_q)$ is a subsequence of $(p_1, \ldots, p_s)$). Since the "1-assignment" of x already justifies the "0-values" of $c_{p'_1}, \ldots, c_{p'_q}$, and since our mapping A_j is monotonous, we obtain $A_i(c \times \{1\})$ by taking entry $A_j(c')$, where $c'_{p'_1} = \ldots = c'_{p'_q} = \hat{0}$, i.e., where $c' = \phi(c)$.
Again, it is not hard to verify the following statements.

Lemma 3.4. *1. The evaluations of (3),(4), and (5) can be carried out in time $O(3^{n_i} n_i)$.*
2. If the mapping A_j is monotonous, then mapping A_i also is monotonous. □

JOIN NODES: Suppose i is a JOIN NODE with children j and k and suppose that $X_i = X_j = X_k = (x_{i_1}, \ldots, x_{i_{n_i}})$. Let $c = (c_1, \ldots, c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$ be a coloring for X_i. We say that $c' = (c'_1, \ldots, c'_{n_i})$, $c'' = (c''_1, \ldots, c''_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$ *divide* c if

1. $(c_t \in \{1, \hat{0}\} \Rightarrow c'_t = c''_t = c_t)$, and
2. $\big(c_t = 0 \Rightarrow \big[(c'_t, c''_t \in \{0, \hat{0}\}) \wedge (c'_t = 0 \vee c''_t = 0)\big]\big)$.

Then, evaluate the mapping A_i of X_i as follows:

$$\texttt{for all } c \in \{0, \hat{0}, 1\}^{n_i} \texttt{ do}$$
$$A_i(c) \;\leftarrow\; \min\{A_j(c') + A_k(c'') - \#_1(c) \mid c' \text{ and } c'' \text{ divide } c\} \qquad (6)$$

In other words, in order to determine the value $A_i(c)$, we look up the corresponding values for coloring c in A_j (which gives us the minimum dominating set for c needed for the bags considered up to this stage in the left subtree) and in A_k (the minimum dominating set for c needed according to the right subtree), add the corresponding values, and subtract the number of "1-assignments" in c, since they would be counted twice, otherwise.
Clearly, if coloring c of node i assigns the colors 1 or $\hat{0}$ to a vertex x in X_i, we have to make sure that we use colorings c' and c'' of the children j and k that also assign the same color to x. However, if c assigns color 0 to x, it is sufficient to justify this color by *at least one* of the colorings c' or c''. Observe that, by the monotonicity of A_j and A_k we obtain the same "min" in assignment (6), by replacing condition 2. in the definition of "divide" by:

$$2'.\ \big(c_t = 0 \;\Rightarrow\; (c'_t, c''_t \in \{0, \hat{0}\} \wedge c'_t \neq c''_t)\big).$$

This observation will be the key to prove the following statement on the running time for this step.

Lemma 3.5. *1. The evaluations in (6) can be carried out in time $O(4^{n_i})$.*
2. If the mappings A_j and A_k are monotonous, then mapping A_i also is monotonous.

Proof. The second statement basically follows from the definition of "divide." As to the time complexity, note that the running time of this step is given by

$$\sum_{c \in \{0,\hat{0},1\}^{n_i}} \big|\{(c', c'') \mid c' \text{ and } c'' \text{ divide } c\}\big|. \qquad (7)$$

For given $c \in \{0, \hat{0}, 1\}^{n_i}$, with $z := \#_0(c)$, we have 2^z many pairs (c', c'') that divide c (if we use condition 2'. instead of 2. (sic!)). Since there are $2^{n_i - z}\binom{n_i}{z}$ many colorings c with $\#_0(c) = z$, again using condition 2'. instead of 2., the expression in (7) equates to

$$\sum_{z=0}^{n_i} 2^{n_i - z} \binom{n_i}{z} \cdot 2^z \quad = \quad 4^{n_i}.$$

□

Step 3 (final evaluation). Let r denote the root of T. For the domination number $\gamma(G)$, we finally get

$$\gamma(G) = \min\{A_r(c) \mid c \in \{0, 1\}^{n_r}\}. \qquad (8)$$

The minimum in Equation (8) is taken only over colorings containing colors 1 and 0, since a valid dominating set does not contain "unresolved" vertices of color $\hat{0}$. Also, note that, when bookkeeping how the minima in the assignments (2), (6), and (8) of Step 2 and Step 3 were obtained, this algorithm constructs a dominating set D corresponding to $\gamma(G)$.

3.3 Correctness and Time Complexity

For the correctness of the algorithm, we observe the following. Firstly, property 1. of a tree decomposition (see Definition 2.1) guarantees that each vertex is assigned a color. Secondly, in our initialization Step 1, as well as in the updating process for INTRODUCE NODES and JOIN NODES of Step 2, we made sure that the assignment of color 0 to a vertex x always guarantees that, at the current stage of the algorithm, x is already dominated by a vertex from previous bags. Since, by property 2. of a tree decomposition (see Definition 2.1), any pair of neighbors appears in at least one bag, the validity of the colorings was checked for each such pair of neighbors. And, thirdly, property 3. of a tree decomposition (see Definition 2.1), together with the comments given in Step 2 of the algorithm, implies that the updating of each mapping is done consistently with all mappings that have been visited earlier in the algorithm.

Lemma 3.6. *The total running time of the algorithm is $O(4^k n)$.*

Proof. This follows directly from Lemmas 3.2, 3.3, 3.4, and 3.5. □

This finishes the proof for Theorem 3.1.

We remark that Aspvall *et al.* [4] addressed the memory requirement problem arising in the type of algorithms described above.

4 Further Applications and Extensions

In this section, we describe how our new dynamic programming strategy can be applied to further "domination-like" problems as, e.g., treated in [19–21].

4.1 Dominating Set with Property P

In the following, a *property* P of a vertex set $V' \subseteq V$ of an undirected graph $G = (V, E)$ will be a Boolean predicate which yields true or false values when given as input V, E, and V'. Since V and E will always be clear from the context, we will simply write $P(V')$ instead of $P(V, E, V')$.

The DOMINATING SET WITH PROPERTY P problem is the task to find a minimum size dominating set D with property P, i.e., such that $P(D)$ is true.

Examples for such problems (all also appearing in [20, 21]) are:

- the INDEPENDENT DOMINATING SET problem, where the property $P(D)$ of the dominating set D is that D is independent,

- the TOTAL DOMINATING SET problem, where the property $P(D)$ of the dominating set D is that each vertex of D has a neighbor in D,
- the PERFECT DOMINATING SET problem, where the property $P(D)$ is that each vertex which is not in D has *exactly* one neighbor in D,
- the PERFECT INDEPENDENT DOMINATING SET problem, also known as the PERFECT CODE problem, where the dominating set has to be perfect and independent, and
- the TOTAL PERFECT DOMINATING SET problem, where the dominating set has to be total and perfect.

For all these instances, we obtain algorithms where the base q_i of the exponential term and the number λ_i of colors needed for the mappings in the dynamic programming are as follows:

Theorem 4.1. *If a width k and n nodes tree decomposition of a graph is known, then we can solve the subsequent problems $\mathcal{P}_i$ in time $O(q_i^k n)$, using λ_i colors in the dynamic programming step:*

- $\mathcal{P}_1 =$ INDEPENDENT DOMINATING SET*:* $q_1 = 4$, $\lambda_1 = 3$*;*
- $\mathcal{P}_2 =$ TOTAL DOMINATING SET*:* $q_2 = 5$, $\lambda_2 = 4$*;*
- $\mathcal{P}_3 =$ PERFECT DOMINATING SET*:* $q_3 = 4$, $\lambda_3 = 3$*;*
- $\mathcal{P}_4 =$ PERFECT CODE*:* $q_4 = 4$, $\lambda_4 = 3$*;*
- $\mathcal{P}_5 =$ TOTAL PERFECT DOMINATING SET*:* $q_5 = 5$, $\lambda_5 = 4$.

Proof. (Sketch) For problem $\mathcal{P}_1$, in contrast to the algorithm given in the proof of Theorem 3.1 (see Subsection 3.2), after each update of a mapping A_i for bag X_i, we check, for each coloring $c \in \{0, \hat{0}, 1\}^{n_i}$, if there exist two vertices $x, y \in X_i$ that both are assigned color 1 by c, and, if so, set $A_i(c) \leftarrow +\infty$.

For problem $\mathcal{P}_2$, one must also distinguish for the vertices in the domination set whether or not they have been dominated by other vertices from the dominating set. We may use 4 colors:

- 1, meaning that the vertex is in the dominating set and it is already dominated;
- $\hat{1}$, meaning that the vertex is in the dominating set and it still needs to be dominated;
- 0, meaning that the vertex is not in the dominating set and it is already dominated;
- $\hat{0}$, meaning that the vertex is not in the dominating set and it still needs to be dominated.

The partial ordering $\prec$ on $C := \{0, \hat{0}, 1, \hat{1}\}$, according to which our mappings will be monotonous, is given by $\hat{1} \prec 1$, $\hat{0} \prec 0$, and $d \prec d$ for all $d \in C$.

The various steps of the algorithm for updating the mappings are similar the ones given in the algorithm of Theorem 3.1 (see Subsection 3.2). The most cost-expensive part again is a JOIN NODE. Here, in the assignment (6), we have to adapt the definition of "divide" as follows: For a coloring $c = (c_1, \ldots, c_{n_i}) \in \{0, \hat{0}, 1, \hat{1}\}^{n_i}$ for X_i, we say that the two colorings $c' = (c'_1, \ldots, c'_{n_i})$, $c'' = (c''_1, \ldots, c''_{n_i}) \in \{0, \hat{0}, 1, \hat{1}\}^{n_i}$ *divide* c if

1. $\big(c_t = 0 \;\Rightarrow\; (c'_t, c''_t \in \{0, \hat{0}\} \wedge c'_t \neq c''_t)\big)$, and
2. $\big(c_t = 1 \;\Rightarrow\; (c'_t, c''_t \in \{1, \hat{1}\} \wedge c'_t \neq c''_t)\big)$.

Similar to the proof of Lemma 3.5 the running time for updating a JOIN NODE is given by

$$\sum_{c \in \{0,\hat{0},1,\hat{1}\}^{n_i}} \big|\{(c', c'') \mid c' \text{ and } c'' \text{ divide } c\}\big|. \tag{9}$$

We use a combinatorial argument to compute this expression. For a fixed coloring $c \in \{0, \hat{0}, 1, \hat{1}\}^{n_i}$, we have $\#_0(c) \in \{0, \ldots, n_i\}$, and $\#_1(c) \in \{0, \ldots, n_i - \#_0(c)\}$. The number of colorings $c \in \{0, \hat{0}, 1, \hat{1}\}^{n_i}$ with $\#_0(c) = z_0$ and $\#_1(c) = z_1$ is given by $\binom{n_i}{z_0}\binom{n_i - z_0}{z_1}$. Since, by definition of "divide," for each position in c with $c_t = 0$ or $c_t = 1$, we have to consider two different divide pairs, we get

$$\begin{aligned}
&\sum_{c \in \{0,\hat{0},1,\hat{1}\}^{n_i}} \big|\{(c', c'') \mid c' \text{ and } c'' \text{ divide } c\}\big| \\
&= \sum_{\#_0(c)=0}^{n_i} \sum_{\#_1(c)=0}^{n_i - \#_0(c)} \binom{n_i}{\#_0(c)} \binom{n_i - \#_0(c)}{\#_1(c)} 2^{\#_0(c)}\, 2^{\#_1(c)} \\
&= \sum_{\#_0(c)=0}^{n_i} \binom{n_i}{\#_0(c)} 2^{\#_0(c)}\, 3^{n_i - \#_0(c)} \qquad = \quad 5^{n_i}
\end{aligned}$$

This determines the running time of the algorithm.

For problem $\mathcal{P}_3$, in contrast to the algorithm given in the proof of Theorem 3.1 (see Subsection 3.2), a vertex is colored "grey" if it is dominated by exactly one "black" vertex which either lies in the "current" bag of the tree decomposition algorithm or in one of its child bags.

For problem $\mathcal{P}_4$ we can use appropriate combinations of the arguments given for problems $\mathcal{P}_1$, $\mathcal{P}_3$.

For problem $\mathcal{P}_5$ we can use appropriate combinations of the arguments given for problems $\mathcal{P}_2$, $\mathcal{P}_3$. □

Note that our updating technique which makes strong use of the monotonicity of the mappings involved yields a basis in the exponential term of the running time which outperforms the results of Telle and Proskurowski. The corresponding constants q'_i for the above listed problems that were derived in [20, Theorem 4, Table 1], and [21, Theorem 5.7] are $q'_1 = 9$, $q'_2 = 16$, $q'_3 = 9$, $q'_4 = 9$, and $q'_5 = 16$. (See Table 2 for an overview.)

4.2 Weighted Versions of Dominating Set

Our algorithm can be adapted to the weighted version of DOMINATING SET (and its variants): Take a graph $G = (V, E)$ together with a positive integer weight function $w : V \to \mathbb{N}$. The weight of a vertex set $D \subseteq V$ is defined as

$w(D) = \sum_{v \in D} w(v)$. The WEIGHTED DOMINATING SET problem is the task to determine, given a graph $G = (V, E)$ and a weight function $w : V \to \mathbb{N}$, a dominating set with minimum weight.

Only small modifications in the bookkeeping technique used in Theorem 3.1 (or Theorem 4.1) are necessary in order to solve the weighted version of DOMINATING SET (and its variations). More precisely, we have to adapt the initialization (1) of the mappings A_i for the bag $X_i = (x_{i_1}, \ldots, x_{i_{n_i}})$ according to:

$$\texttt{for all } c = (c_1, \ldots, c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i} \texttt{ do}$$
$$A_i(c) \leftarrow \begin{cases} +\infty \text{ if } c \text{ is locally invalid for } X_i \\ w(c) \text{ otherwise,} \end{cases} \tag{10}$$

where $w(c) := \sum_{t=0, c_t=1}^{n_i} w(x_{i_t})$. The updating of the mappings A_i in Step 2 in the algorithm of Theorem 3.1 (or Theorem 4.1) is adapted similarly.

4.3 Red-Blue Dominating Set

We finally turn our attention to the following version of DOMINATING SET, called RED-BLUE DOMINATING SET[5] ([11, Exercise 3.1.5]):

An instance of RED-BLUE DOMINATING SET is given by a (planar) bipartite graph $G = (V, E)$, where the bipartition is given by $V = V_{red} \cup V_{blue}$. The question is to determine a set $V' \subseteq V_{red}$ of minimum size such that every vertex of V_{blue} is adjacent to at least one vertex of V'.

This problem is directly related to the FACE COVER problem (see [5, 11]). A *face cover* C of an undirected plane graph $G = (V, E)$ (i.e., a planar graph with a fixed embedding) is a set of faces that cover all vertices of G, i.e., for every vertex $v \in V$, there exists a face $f \in C$ so that v lies on the boundary of f. The FACE COVER problem is the task to find a minimum size face cover for a given plane graph.

The relation between FACE COVER and RED-BLUE DOMINATING SET is as follows. For an instance $G = (V, E)$ of the FACE COVER problem, consider the following graph: Add a vertex to each face of G, and make each such "face vertex" adjacent to all vertices that are on the boundary of that face. These are the only edges of the bipartite graph $G' = (V', E')$. Write $V' = V \cup V_F$, where V_F is the set of face vertices, i.e., each $v \in V_F$ represents a face f_v in G. In other words, V and V_F form the bipartition of G'. Observe that G' can be viewed as an instance of RED-BLUE DOMINATING SET.

Theorem 4.2. *Let a bipartite graph $G = (V, E)$ with bipartition $V = V_{red} \cup V_{blue}$ be given together with a tree decomposition of width k. Then,* RED-BLUE DOMINATING SET *can be solved in time $O(3^k n)$, where n is the number of nodes of the tree decomposition.*

[5] Observe that RED-BLUE DOMINATING SET is *not* a variant of DOMINATING SET in the sense of the first subsection, because a solution V' is not a dominating set, since red vertices cannot and hence need not be dominated by red vertices.

Proof. (Sketch) Basically, the technique exhibited in Theorem 3.1 (see Subsection 3.2) can be applied. Due to the bipartite nature of the graph, only two "states" have to be stored for each vertex: red vertices are either within the dominating set or not (represented by colors 1_{red} and 0_{red}, respectively), and blue vertices are either already dominated or not yet dominated (represented by colors 0_{blue} and $\hat{0}_{\text{blue}}$, respectively).

We consider our bags as bipartite sets, i.e.,

$$X_i := X_{i,\text{red}} \cup X_{i,\text{blue}},$$

where $X_{i,\text{red}} := X_i \cap V_{\text{red}}$ and $X_{i,\text{blue}} := X_i \cap V_{\text{blue}}$. Let $n_{i,\text{red}} := |X_{i,\text{red}}|$ and $n_{i,\text{blue}} := |X_{i,\text{blue}}|$, i.e., $|X_i| =: n_i = n_{i,\text{red}} + n_{i,\text{blue}}$.

The partial ordering $\prec$ on the color set $C = C_{\text{red}} \cup C_{\text{blue}}$, where $C_{\text{red}} := \{1_{\text{red}}, 0_{\text{red}}\}$ and $C_{\text{blue}} := \{0_{\text{blue}}, \hat{0}_{\text{blue}}\}$, is given by $\hat{0}_{\text{blue}} \prec 0_{\text{blue}}$ and $d \prec d$ for all $d \in C$.

A *valid* coloring for X_i is a coloring where we assign colors from C_{red} to vertices in $X_{i,\text{red}}$ and colors from C_{blue} to vertices in $X_{i,\text{blue}}$. The various steps of the algorithm for updating the mappings are similar to the ones given in the algorithm of Theorem 3.1 (see Subsection 3.2).

Again, the most cost-expensive part is the updating of a JOIN NODE. For a correct updating of JOIN NODES, we adapt the definition of "divide" that appears in the assignment (6) according to: For a valid coloring $c = (c_1, \ldots, c_{n_i}) \in C^{n_i}$ of X_i, we say that the valid colorings $c' = (c'_1, \ldots, c'_{n_i})$, $c'' = (c''_1, \ldots, c''_{n_i}) \in C^{n_i}$ *divide* c if

1. $\big(c_t \neq 0_{\text{blue}} \;\Rightarrow\; (c'_t, c''_t = c_t)\big)$, and
2. $\big(c_t = 0_{\text{blue}} \;\Rightarrow\; (c'_t, c''_t \in \{0_{\text{blue}}, \hat{0}_{\text{blue}}\} \wedge c'_t \neq c''_t)\big)$.

For a fixed valid coloring c that contains $z := \#_{0_{\text{blue}}}(c)$ many colors 0_{blue}, the number of pairs that divide c is 2^z. Since there are $2^{n_{i,\text{red}}}\binom{n_{i,\text{blue}}}{z}$ many colorings with $\#_{0_{\text{blue}}}(c) = z$, the total number of pairs that divide a fixed coloring c is upper-bounded by

$$\sum_{z=0}^{n_{i,\text{blue}}} 2^{n_{i,\text{red}}} \binom{n_{i,\text{blue}}}{z} \cdot 2^z \quad = \quad 2^{n_{i,\text{red}}} \cdot 3^{n_{i,\text{blue}}} \quad \leq \quad 3^{n_i}.$$

This determines the running time of the algorithm. Note that in the worst case, for a bag X_i, we may have $n_{i,\text{red}} = 0$ and $n_{i,\text{blue}} = n_i$. □

5 Conclusion

In this paper, we focused on solving domination-like problems for graphs that are given together with a tree decomposition of width k. We presented a new "monotonicity" argument for the usual dynamic programming procedure. This

Problem	λ	q	q'
(WEIGHTED) DOMINATING SET	3	4	9
INDEPENDENT DOMINATING SET	3	4	9
PERFECT DOMINATING SET	3	4	9
PERFECT CODE	3	4	9
TOTAL DOMINATING SET	4	5	16
TOTAL PERFECT DOMINATING SET	4	5	16
RED-BLUE DOMINATING SET	4	4	-
VERTEX COVER	2	2	4
INDEPENDENT SET	2	2	4

Table 2. Summary of our results (Theorems 3.1, 4.1, 4.2) **and comparison with previous work**. The entries in the second column give the number λ of colors used in our dynamic programming step. The third column gives the base q for our $O(q^k n)$ time algorithm (k being the width of the given tree decomposition). The entries of the fourth column give the corresponding base values q' of the so far best known algorithms by Telle and Proskurowski [20, Theorem 4, Table 1], and [21, Theorem 5.7]. The results for VERTEX COVER and INDEPENDENT SET can be obtained by a strategy similar to the one for DOMINATING SET. However, in contrast to the other problems, the up-dating process is straight forward and much less involved.

new strategy yields significant improvements over the so far best known algorithms. The results, and the corresponding improvements over previous work are summarized in Table 2.

Observe that we obtained "base values" q which are at most $\lambda + 1$, where λ denotes the number of colors needed in the dynamic programming. The corresponding base values q' obtained by Telle and Proskurowski all are of the form $q' = \lambda^2$ (see Table 2).

It remains a challenge for future research whether the "base values" q in Table 2 all can be lowered to *match* the corresponding value λ of colors needed in the dynamic programming. For instance, in the case of DOMINATING SET we only need three colors in the dynamic programming tables, but the updating process so far invokes the exponential term 4^k. Such an improvement would imply a significant speed-up of our software package described in [3].

Acknowledgments. We profited from constructive and inspiring collaboration with Hans L. Bodlaender and Henning Fernau. We are grateful to Frederic Dorn for his careful (and successful) work on implementing the presented algorithms. We thank Ute Schmid for her helpful hints concerning some combinatorial arguments.

References

1. J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proceedings 7th SWAT 2000*, Springer-Verlag LNCS 1851, pp. 97–110, 2000.
2. J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. In *Proceedings 28th ICALP 2001*, Springer-Verlag LNCS 2076, pp. 261–272, 2001.
3. J. Alber, F. Dorn, and R. Niedermeier. Experiments on optimally solving parameterized problems on planar graphs. Manuscript, December 2001.
4. B. Aspvall, A. Proskurowski, and J. A. Telle. Memory requirements for table computations in partial k-tree algorithms. *Algorithmica* **27**: 382–394, 2000.
5. D. Bienstock and C. L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM J. Comput.* **17**:53–76, 1988.
6. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.* **25**:1305–1317, 1996.
7. H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings 22nd MFCS'97*, Springer-Verlag LNCS 1295, pp. 19–36, 1997.
8. A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: a Survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 1999.
9. J. Chen, I. Kanj, and W. Jia. Vertex cover: further observations and further improvements. In *Proceedings 25th WG*, Springer-Verlag LNCS 1665, pp. 313–324, 1999.
10. D. G. Corneil and J. M. Keil. A dynamic programming approach to the dominating set problem on k-trees. *SIAM J. Alg. Disc. Meth.*, **8**: 535–543, 1987.
11. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer-Verlag, 1999.
12. T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs*. Monographs and textbooks in Pure and Applied Mathematics Vol. 208, Marcel Dekker, 1998.
13. T. W. Haynes, S. T. Hedetniemi, and P. J. Slater (eds.). *Domination in Graphs; Advanced Topics*. Monographs and textbooks in Pure and Applied Mathematics Vol. 209, Marcel Dekker, 1998.
14. T. Kloks. *Treewidth. Computations and Approximations*. Springer-Verlag LNCS 842, 1994.
15. A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. Hoesel. Treewidth: Computational Experiments. *Electronic Notes in Discrete Mathematics 8*, Elsevier Science Publishers, 2001.
16. D. Kratsch. Algorithms. Chapter 8 in [13].
17. K. Mehlhorn and S. Näher. *LEDA: A Platform of Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, England, 1999.
18. R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In *Proc. 16th STACS'99*, Springer-Verlag LNCS 1563, pp. 561–570, 1999.
19. J. A. Telle. Complexity of domination-type problems in graphs. *Nordic J. Comput.* **1**:157–171, 1994.
20. J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proceedings 3rd WADS'93*, Springer-Verlag LNCS 709, pp. 610–621, 1993.
21. J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discr. Math.* **10(4)**:529–550, 1997.