Algorithmica (2011) 61:252–273
DOI 10.1007/s00453-010-9418-9

Branch and Recharge: Exact Algorithms for Generalized Domination

Fedor V. Fomin · Petr A. Golovach · Jan Kratochvíl · Dieter Kratsch · Mathieu Liedloff

Received: 17 September 2009 / Accepted: 20 May 2010 / Published online: 4 June 2010
© Springer Science+Business Media, LLC 2010

Abstract In this paper we present branching algorithms for infinite classes of problems.

The novelty in the design and analysis of our branching algorithms lies in the fact that the weights are redistributed over the graph by the algorithms. Our particular setting to make this idea work is a combination of a branching approach with a recharging mechanism. We call it Branch & Recharge. To demonstrate this approach we consider a generalized domination problem.

A preliminary version of the paper appeared in the proceedings of WADS 2007 [7].

Research of J. Kratochvíl was supported by Czech research projects MSM0021620838 and 1M0545.

F.V. Fomin
Department of Informatics, University of Bergen, 5020 Bergen, Norway
e-mail: fedor.fomin@ii.uib.no

P.A. Golovach
School of Engineering and Computing Sciences, Durham University, South Road, DH1 3LE Durham, UK
e-mail: petr.golovach@durham.ac.uk

J. Kratochvíl
Department of Applied Mathematics, and Institute for Theoretical Computer Science, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
e-mail: honza@kam.ms.mff.cuni.cz

D. Kratsch (✉)
Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine—Metz, 57045 Metz Cedex 01, France
e-mail: kratsch@univ-metz.fr

M. Liedloff
Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, 45067 Orléans Cedex 2, France
e-mail: liedloff@univ-orleans.fr

Let σ and ϱ be two nonempty sets of nonnegative integers. A vertex subset $S \subseteq V$ of an undirected graph $G = (V(G), E(G))$ is called a (σ, ϱ)-dominating set of G if $|N(v) \cap S| \in \sigma$ for all $v \in S$ and $|N(v) \cap S| \in \varrho$ for all $v \in V \setminus S$. This notion generalizes many domination-type graph invariants.

We present Branch & Recharge algorithms enumerating all (σ, ϱ)-dominating sets of an input graph G in time $O^*(c^n)$ for some $c < 2$, if σ is successor-free, i.e., it does not contain two consecutive integers, and either both σ and ϱ are finite, or one of them is finite and $\sigma \cap \varrho = \emptyset$. Our algorithm implies a non trivial upper bound of $O^*(c^n)$ on the number of (σ, ϱ)-dominating sets in an n-vertex graph under the above conditions on σ and ϱ.

Keywords Exact exponential algorithms · NP-hard problems · Generalized domination · Branch and recharge

1 Introduction

Exact exponential time algorithms solving NP-hard problems have been studied extensively in the last decade. Dynamic Programming, Inclusion-Exclusion and Branch & Reduce are the most important techniques to design and analyze such algorithms. For an introduction to exact algorithms we refer to Woeginger's survey [25]. Many exact exponential time algorithms for NP-hard problems are branching algorithms. By now the use of sophisticated measures when analyzing the running time of branching algorithms is a well-established technique which is often called Measure & Conquer (see e.g. [1, 6, 8, 11, 12]).

The typical branching algorithm consists of some branching and reduction rules. Usually the correctness of the algorithm is straightforward, but its running time analysis not. Very often to improve the analysis of the algorithm the following approach is used. We assign different weights to the input elements according to some rules. Say the input is a graph. Then we can assign a weight to each vertex of the graph and introduce the measure of the graph as the sum of the weights of all its vertices. One may normalize the measure such that all weights are reals of [0, 1], and thus every instance of a subproblem generated by the branching algorithm has a non negative measure at most $|V| = n$, i.e. the number of vertices of the input graph. Let us emphasize that the measure is a tool to analyze the running time. There is neither weight nor measure in the description of such algorithms. The usage of Measure & Conquer is in the analyses of algorithms. It was shown in [11] that introducing measure in the analyses of branching algorithm can lead to substantial improvements in the time analysis. Indeed Measure & Conquer is one of the best methods available to analyze branching algorithms.

Given the power of measures and weights when analyzing branching algorithms, why do we use measures and weights only to support running time analysis? Why should it not be possible to incorporate the weights in the branching algorithm? This natural idea was our original motivation.

The main goal of this paper is to introduce the new approach which we call Branch & Recharge. Like in Measure & Conquer, weights are assigned to the vertices of the

Table 1 Examples of (σ, ϱ)-dominating sets. Here $\mathbb{N}$ denotes the set of positive integers and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Our Branch & Recharge enumeration algorithms apply to the lower four problems

σ	ϱ	(σ, ϱ)-dominating set
$\mathbb{N}_0$	$\mathbb{N}$	dominating set
$\{0\}$	$\mathbb{N}_0$	independent set
$\{0\}$	$\{1\}$	1-perfect code
$\{0\}$	$\{0, 1\}$	strong stable set
$\{0\}$	$\mathbb{N}$	independent dominating set
$\{1\}$	$\{1\}$	total perfect dominating set

input graph. Branching is essentially done in a simple way such that the measure of an instance obtained by branching is smaller than the measure of the original instance. This decrease is used to obtain branching vectors and running time in a simple way. The major novelty is a redistribution of the weights called recharging, done whenever it becomes necessary.

We demonstrate the Branch & Recharge approach by applying it to generalized domination problems.

1.1 Generalized Domination

We consider finite undirected graphs without loops or multiple edges. Thus a graph is a pair $G = (V(G), E(G))$ where $V(G)$ is the (finite) set of vertices and $E(G)$ the set of edges. The size of G is the number of vertices, and throughout the paper we reserve $n = |V(G)|$ to denote this quantity. We call two vertices u, v *adjacent* if they form an edge, i.e., if $uv \in E(G)$. The *open neighborhood* of a vertex $u \in V$ is the set of the vertices adjacent to it, denoted by $N(u) = \{x : xu \in E\}$. A set of vertices $S \subseteq V(G)$ is *dominating* if every vertex of G is either in S or adjacent to a vertex in S. Finding a dominating set of the smallest possible size belongs to the most important optimization problems on graphs. Many generalizations have been studied, such as independent dominating set, connected dominating set, efficient dominating set, etc. (cf. [18]).

In [24], Telle introduced the following framework of domination-type graph invariants (see also [17, 19]). Let σ and ϱ be two nonempty sets of nonnegative integers. A vertex subset $S \subseteq V(G)$ of an undirected graph $G = (V(G), E(G))$ is called a (σ, ϱ)-*dominating set* of G if $|N(v) \cap S| \in \sigma$ for all $v \in S$ and $|N(v) \cap S| \in \varrho$ for all $v \in V \setminus S$. Table 1 shows a sample of previously defined and studied graph invariants which can be expressed in this framework.

When studying algorithmic complexity of problems on (σ, ϱ)-dominating sets the following decision, search and counting problems are of interest.

$\exists(\sigma, \varrho)$-DS: Does an input graph G contain a (σ, ϱ)-dominating set?
ENUM-(σ, ϱ)-DS: Given a graph G, list all (σ, ϱ)-dominating sets of G.
#-(σ, ϱ)-DS: Given a graph G, determine the number of (σ, ϱ)-dominating sets of G.
MAX-(σ, ϱ)-DS: Given a graph G, find a (σ, ϱ)-dominating set of maximum size.
MIN-(σ, ϱ)-DS: Given a graph G, find a (σ, ϱ)-dominating set of minimum size.

Obviously, the enumeration problem ENUM-(σ, ϱ)-DS is the most difficult one, since as soon as we have all (σ, ϱ)-dominating sets in a list, we can quickly see if the list is nonempty (and hence answer the $\exists(\sigma, \varrho)$-DS problem), we can also compare the sizes of the listed sets to answer the minimization and maximization questions, and we can quickly count the number of listed sets. However, maybe slightly surprisingly, already the existence problem is NP-complete for many parameter pairs σ and ϱ, including some of those listed in Table 1 (1-perfect code and total perfect dominating set). In fact, Telle [24] proves that $\exists(\sigma, \varrho)$-DS is NP-complete for every two finite nonempty sets σ, ϱ such that $0 \notin \varrho$. Complexity results on partitioning graphs into (σ, ϱ)-dominating sets can be found in [19]. More complexity results on $\exists(\sigma, \varrho)$-DS restricted to special graph classes and also from the fixed parameter complexity point of view are presented in [13–15]. Exact algorithms for generalized domination are given in [10].

The main contribution of the paper is the presentation of a new approach to design and analyze branching algorithms. The novelty is that weights are redistributed over the graph. We call this approach Branch & Recharge. It may also be seen as combination of a *branching* algorithm with a *recharging* mechanism which was inspired by recharging techniques in proofs of graph coloring theorems.

The application of the new method to generalized domination problems establishes two main results. Firstly, we present a $O^*(c^n)$-time[1] algorithm for the ENUM-(σ, ϱ)-DS problem, where the constant $c < 2$ depends on σ and ϱ, for a fairly wide class of parameter sets σ and ϱ. Secondly, an upper bound on the running time of an enumeration algorithm immediately implies an upper bound on the number of enumerated objects. Thus our Branch & Recharge algorithm has the combinatorial corollary stating that every isolate-free graph with n vertices contains at most $O^*(c^n)$ (σ, ϱ)-dominating sets (under the same assumptions on σ and ϱ).

The relation of exact exponential time algorithms and combinatorial upper bounds is interesting in its own. For several moderately exponential-time algorithms, the running time analysis is based on combinatorial theorems bounding the number of certain objects. For example a number of coloring algorithms are based on bounds on the number of certain maximal independent sets and bipartite subgraphs in a graph [3–5, 21] and the algorithm for domatic number in [2] is based on a bound for the number of minimal dominating sets.

From the other side, the time analysis of a branching algorithm might provide the proof of combinatorial upper bounds. The most famous combinatorial result of this type is the well-known Moon-Moser theorem stating that the maximum number of maximal cliques (resp. maximal independent sets) of an n-vertex graph is $3^{n/3}$ [22] (while its original proof is combinatorial it can be easily turned into a branching algorithm enumerating all maximal independent sets). Techniques inspired by the analysis of exact algorithms were later used to obtain the bounds on the number of minimal dominating sets, minimal feedback vertex sets, and maximal r-regular subgraphs among others [8, 9, 16]. In general, moderately exponential-time algorithms

[1] As has recently become standard, we write $f(n) = O^*(g(n))$ if $f(n) \leq p(n) \cdot g(n)$ for some polynomial $p(n)$.

and their analysis seem to be useful tools to obtain such combinatorial results (up to a polynomial factor).

Our paper is organized as follows. In Sect. 2 we present basic observations on (σ, ϱ)-dominating sets and also discuss our conditions imposed on the parameter sets σ and ϱ. In Sect. 3 our Branch & Recharge algorithms are presented and their running time analysis is given. Combinatorial upper and lower bounds on the number of (σ, ϱ)-dominating sets are discussed in Sect. 4.

2 Preliminaries and the Main Combinatorial Theorem

In this paper we present Branch & Recharge algorithms enumerating all (σ, ϱ)-dominating sets in an n-vertex graph. A trivial brute-force algorithm trying all possible subsets and checking (we assume here that the check whether an integer is in σ or ρ can be performed in polynomial time) if a given subset is a (σ, ϱ)-dominating set, works in time $O^*(2^n)$. It is easy to show that in general case, there is no enumeration algorithm better than $O^*(2^n)$. This is because a graph on n vertices may contain as many as 2^n (σ, ϱ)-dominating sets, e.g., if $0 \in \sigma \cap \varrho$, then the edgeless graph does. Another less trivial example is $\sigma = \varrho = \{0, 1, \ldots, d\}$, since then any set of vertices in a graph of maximum degree d is a (σ, ϱ)-dominating. Therefore, no enumeration algorithm significantly faster than $\Theta(2^n)$ for the general case of (σ, ϱ)-domination is possible.

Before discussing the choice of the conditions on the parameter sets σ and ϱ let us emphasize that our priority is the presentation of Branch & Recharge on interesting problems. Nevertheless we have tried to make the framework work for as large as possible parameter sets σ and ϱ. We call a set of integers *successor-free* if it contains no pair of consecutive integers. In the rest of the paper we use the notation $p = \max \sigma$ and $q = \max \varrho$ (with $p = \infty$ if σ is infinite, and $q = \infty$ if ϱ is infinite). We denote by $\mathbb{N}$ the set of positive integers and by $\mathbb{N}_0$ the set of nonnegative integers.

The crucial technical condition required by our algorithms is the successor-freeness of σ. It is used to make sure that whenever needed recharging is possible. However, simple examples show that even this condition is too general. For example, when σ is the set of even integers and ϱ the set of odd integers, then the complete graph $G = K_n$ contains 2^{n-1} (σ, ϱ)-dominating sets (every odd subset of vertices is one), and yet σ and ϱ are successor-free and disjoint (but both are infinite). Similarly, for $\sigma = \{0\}$ and $\varrho = \mathbb{N}_0$ (σ successor-free and finite, but σ and ϱ are not disjoint), the star $K_{1,n-1}$ contains $2^{n-1} + 1$ (σ, ϱ)-dominating sets.

Another observation concerns disconnected graphs. The number of (σ, ϱ)-dominating sets in a graph is equal to the product of the numbers of (σ, ϱ)-dominating sets in its connected components. Hence it would suffice to consider connected graphs. However, the analysis of our algorithm also works for isolate-free input graphs (i.e., graphs without isolated vertices), which is more interesting for the main combinatorial result of our paper:

Theorem 1 *If σ is successor-free and either both σ and ϱ are finite, or one of them is finite and $\sigma \cap \varrho = \emptyset$, then every isolate-free graph contains at most $O^*(c^n)$ (σ, ϱ)-dominating sets, where $c = c_{\sigma,\varrho} < 2$ is a constant depending on σ and ϱ. Moreover,*

all the (σ, ϱ)*-dominating sets can be enumerated in time* $O^*(c^n)$ (*where* c *is the same constant*).

It is worth noting that the constant c depends only on $p = \max \sigma$ and $q = \max \varrho$. Furthermore we emphasize that the Branch & Recharge algorithms provide explicitly the currently best known constants $c = c_{\sigma,\varrho}$ (and not only the proof that some constant smaller than 2 can be achieved). The theorem is established by the presentation of the Branch & Recharge approach for the corresponding generalized domination problems in the next section. (For some values of $c = c_{\sigma,\varrho} = c_{p,q}$ see also the two tables in the next section.)

Recent results of Gupta et al. [16] give an enumeration algorithm as well as upper and lower bounds on the number of maximal r-regular subgraphs in a given graph. Induced r-regular subgraphs are $(\{r\}, \mathbb{N}_0)$-dominating sets, and thus this result is somewhat related to our work.

Finally we would like to mention that our running time analysis uses standard tools for the worst-case running time analysis of branching algorithms (with very simple branching vectors).

To sketch the main ideas, let us consider a branching rule that, when applied to an instance of weight k, branches into $t \geq 2$ subproblems such that the weight of the instance on subproblem i is at most $k - k_i$, $i \in \{1, 2, \ldots, t\}$, where $k_1, k_2, \ldots, k_t$ are positive reals. Then we say that this branching rule has the *branching vector* $(k_1, k_2, \ldots, k_t)$. The running time of a branching algorithm on an instance of weight k that applies (only) this branching rule can be obtained by solving the linear recurrence

$$T[k] = T[k - k_1] + T[k - k_2] + \cdots + T[k - k_t].$$

Solutions of such linear recurrences can be determined by using the roots of the *characteristic polynomial*

$$x^k - x^{k-k_1} - x^{k-k_2} - \cdots - x^{k-k_t}.$$

In the case of recurrences established by branching algorithms it is known that the characteristic polynomial has a unique positive real root c and that the running time of the branching algorithm is $O^*(c^k)$. This c is also called the branching factor of the branching vector $(k_1, k_2, \ldots, k_t)$.

We refer to a textbook of Discrete Mathematics as e.g. [23] for an introduction to solving linear recurrences. Concerning the use of linear recurrences in the analysis of branching algorithms we refer to [25] for an easy to read introduction and to [20] for a comprehensive presentation.

3 Branch & Recharge Algorithms

The overall presentation of our algorithm will be somewhat different from the one of typical branching algorithms since the changes of the weights are explicitly specified. We do need such a careful algorithm description since details are crucial for the correctness of the algorithm and the time analysis. Though the algorithms do not

need the weights for their execution, the treatment of the weights is crucial for the correctness of the time analysis by recharging. Indeed the correctness proofs are difficult and need a careful analysis of how the weights will change during an execution of the algorithm. On the other hand, the running time analysis becomes easy since it is based on one or two branching vectors only.

We present two Branch & Recharge algorithms (called A and B) depending on conditions on σ and ϱ. First we describe some common and fundamental features of the two Branch & Recharge algorithms. A full description, correctness proofs and time analysis will be given separately in the subsequent subsections.

Case A: σ successor-free, σ and ϱ finite (Sect. 3.1).
Case B: σ successor-free, either σ or ϱ finite, $\sigma \cap \varrho = \emptyset$ (Sect. 3.2).

Let us mention that three of the problems of Table 1 satisfy the conditions of Case A (1-perfect code, strong stable set, total perfect dominating set) and that two of the problems satisfy the conditions of Case B (1-perfect code, independent dominating set). Furthermore, no enumeration algorithm of running time $O^*(c^n)$ with $c < 2$ can be achieved for the problems dominating set and independent set, since a complete graph on n vertices has $2^n - 1$ dominating sets and the star $K_{1,n-1}$ has $2^{n-1} + 1$ independent sets.

The key idea of the Branch & Recharge algorithms is to enumerate all (σ, ϱ) dominating sets by one or two simple branching rules. Using weights on the vertices and operations on the weights, like decreasing and redistributing weights, only one or two branching vectors are needed. This implies a simple running time analysis based on the tools described in the previous section.

Initially every vertex $v \in V(G)$ of the isolate-free input graph $G = (V(G), E(G))$ is assigned weight $w(v) = 1$, and thus the total weight of the input graph is $w(G) = \sum_{v \in V} w(v) = n$. The weight of a vertex will always be a real between 0 and 1, and thus the total weight of any graph treated by the algorithm is at most n.

The Branch & Recharge algorithm recursively builds candidate sets S for (σ, ϱ)-dominating sets in G by calling for a candidate vertex v the procedure **SigmaRho** which consists of three subroutines: **Forcing** (which identifies vertices that must or must not be placed in S), **Recharge** (which prepares the ground for the subroutine **Branch** by sending charges from vertices to other vertices such that the weight of v is recharged to the value 1), and **Branch** (which either selects or discards the candidate vertex v; the core of the algorithm, as it is responsible both for the exponential running time of the algorithm and for the base of the exponential function). All these subroutines and the procedure work with the same graph G and leave it unchanged, and with a global variable L which is the list of candidate sets S. Their parameters are S (containing the vertices selected for the candidate set), $\overline{S}$ (containing the vertices discarded from the candidate set), the weight function $w : V \to [0, 1]$ and an auxiliary directed graph H which is an orientation of a spanning subgraph of G (H is tracking the recharging moves). Moreover, **Forcing**, **Recharge** and **Branch** are called on a particular free vertex v, i.e. v has not been allocated to S or $\overline{S}$ yet.

Vertices belonging to $S \cup \overline{S}$ are called *allocated*. Free vertices keep nonnegative weights. All allocated vertices have weight zero. Once a vertex is allocated in S (we say it is *selected*) or in $\overline{S}$ (we say it is *discarded*) it never changes its status during

further calls. At every stage of the algorithm a vertex is called *free* if it does not belong to $S \cup \overline{S}$.

It should be noted that these algorithms require exponential space (as they are given), but such space is needed only to keep the global variable L which contains the list of all candidate sets S. If we do not need this list then for every candidate set S, we can check whether this is really a (σ, ϱ)-dominating set, and consider it immediately after it has been generated (send it to the output, count it, compare it with current minimum and maximum sets). So the correspondingly modified algorithm uses only polynomial space. Finally we like to mention that no treatment of multiples is needed, since both of our algorithms produce a fixed (σ, ϱ)-dominating set S only once; due to the branching rules used in the algorithms.

In the following two subsections we study Branch & Recharge algorithms for enumerating all (σ, ϱ) dominating sets under certain conditions on the parameter sets σ and ϱ. The Branch & Recharge algorithm solving Case A (Sect. 3.1) is called Algorithm A. It is simpler and its correctness is easier to verify. The Branch & Recharge algorithm solving Case B (Sect. 3.2) is called algorithm B. It shows some new and powerful ideas compared to those used in Algorithm A, in particular a more sophisticated recharging mechanism. Algorithm B illustrates the power and the potentials of the Branch & Recharge paradigm.

3.1 Algorithm A

Throughout this subsection we assume that σ and ϱ are finite, and also that σ is successor-free (Case A). Recall that $p = \max \sigma$ and $q = \max \varrho$. If $\sigma = \varrho = \{0\}$, then for every isolate-free graph the empty set is the unique (σ, ϱ)-dominating set. Hence we may assume that $\max\{p, q\} > 0$.

Before presenting our Branch & Recharge algorithm A and its correctness proof in details, let us sketch some of the main ideas of Branch & Recharge with emphasis on the updates of the weights. The key idea of algorithm A is to guarantee that in any branching step on a chosen vertex v the measure, i.e. the total weight, of the instance decreases by at least 1 when v is discarded, and it decreases by at least $1 + \epsilon$ when v is selected. Here $\epsilon = \frac{1}{\max\{p,q\}} > 0$ is a constant dependent on p and q only. Therefore the branching vector $(1, 1 + \epsilon)$ with $\epsilon > 0$ immediately implies that the running time is $O^*(c^n)$ where c is the branching factor of $(1, 1 + \epsilon)$. Decreasing the total weight is achieved by first recharging the weight of v to 1 if necessary. If v is discarded its weight is set to 0; otherwise (i.e., if v is selected) the weight of v is set to 0 and the weight of some neighbor w of v is decreased by ϵ. (It may help to think of this as if vertex $v \in S$ borrowed an ϵ from its neighbor w.) This demonstrates an advantage of Branch & Recharge: one may use weights to guarantee a certain branching vector.

The idea of the forcing procedure is to allocate vertices or halt (since no solution can be found) if certain obvious conditions are fulfilled. For example, if a free vertex u has negative weight then it has lent more than $\max\{p, q\}$ charges to neighbors in S, and thus it has more than $\max\{p, q\}$ neighbors in S, which is impossible; hence no solution can be found in this instance and the subroutine halts. The forcing is a kind of cleaning procedure that is needed to guarantee that recharging is always possible.

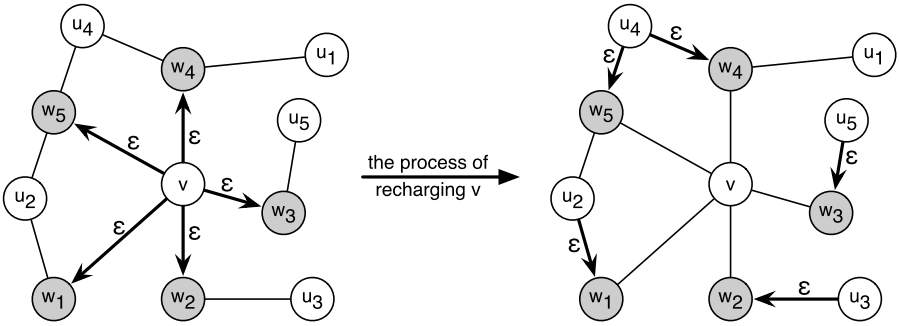

Fig. 1 The process of recharging a vertex v. On the *left* side, a vertex v gave 5ϵ to some neighbors being in S, namely w_1, w_2, w_3, w_4 and w_5. Due to the successor-freeness property of σ, each $w_i, i \in \{1, \ldots, 5\}$, has a free neighbor $u_k, k \in \{1, \ldots, 5\}$ (otherwise, v would be forced by a reduction rule and thus not free when attempting to branch on). The value of ϵ ensures that each free vertex u_k has a weight being no smaller than ϵ (otherwise, such a vertex would have more neighbors in S than allowed by σ and ϱ). On the *right* side, the vertex v is recharged: Instead of giving ϵ's to the w_i's, the ϵ's are given by the u_k's to the w_i's. The redistribution of the weights leads to $w(v) = 1$

Finally recharging is done if the algorithm branches on a vertex v of weight $1 - t\epsilon$ with $t \geq 1$. Hence vertex v lent t charges to its neighbors and all those neighbors $w_1, w_2, \ldots, w_t$ are allocated to S. The idea is to imagine that all those t neighbors w_i of v actually borrowed ϵ from a wrong vertex. Now we imagine that each such neighbor w_i of v borrows from another neighbor u_i (instead of v) and passes this charge to v. This is the underlying idea of recharging in algorithm A. A proof that this recharging is always possible needs a more careful study of the weights in instances of a subproblem and will be given later.

The correctness proof of our Branch & Recharge algorithm needs a detailed description of algorithm A, thus we describe it in pseudocode (the global variable L and the input graph G are not listed in the preamble). Note that whenever a charge is sent from one vertex to another along an edge of H, its value is equal to ϵ. We shall prove later that the outdegree of a vertex in H is at most $\max\{p, q\}$, and thus no vertex ever gets a negative weight.

Algorithm Main-EnumSigmaRho-A(G)
Prepossessing: Choose an arbitrary vertex v_1 and order the vertex set of G in a BFS ordering $B : v_1, v_2, \ldots, v_n$
Initialization: $L := \emptyset$; $S := \emptyset$; $\overline{S} := \emptyset$; $H := (V(G), \emptyset)$; forall $v \in V(G)$ do $w(v) := 1$
SigmaRhoA($S, \overline{S}, w, H, B$)
Termination:
forall $S \in L$ do
  if S is not a (σ, ϱ)-dominating set in G then $L := L \setminus \{S\}$
output(L)

Procedure SigmaRhoA$(S, \overline{S}, w, H, B)$
if there is no free vertex then $L := L \cup \{S\}$
else
let v be the last free vertex in the BFS ordering B of $V(G)$
if $v = v_1$ then
$L := L \cup \{S, S \cup \{v\}\}$
else
ForcingA$(v, S, \overline{S}, w, H, B)$
if ForcingA halted then Halt
if v is still free then
RechargeA$(v, S, \overline{S}, w, H, B)$
BranchA$(v, S, \overline{S}, w, H, B)$
else SigmaRhoA$(S, \overline{S}, w, H, B)$

Subroutine ForcingA$(v, S, \overline{S}, w, H, B)$
if $\exists$ free vertex x s.t. x is adjacent to the free vertex v in G and
$|N(x) \cap S| = \max\{p, q\}$ then
$\overline{S} := \overline{S} \cup \{v\}, w(v) := 0$
else
if $\exists x \in S$ s.t. v is its unique free neighbor in G then
case
$|N(x) \cap S| \in \sigma$ then $\overline{S} := \overline{S} \cup \{v\}, w(v) := 0$
$|N(x) \cap S| + 1 \in \sigma$ then $S := S \cup \{v\}, w(v) := 0$
$\{|N(x) \cap S|, |N(x) \cap S| + 1\} \cap \sigma = \emptyset$ then Halt
if $\exists x$ s.t. $|N(x) \cap S| > \max\{p, q\}$ then Halt

Subroutine RechargeA$(v, S, \overline{S}, w, H, B)$
if $w(v) < 1$ then
let $\{w_1, \ldots, w_t\} = \{x : vx \in E(H)\}$
for $i := 1$ to t do let $u_i \neq v$ be a free neighbor (in G) of w_i
for $i := 1$ to t do
$w(u_i) := w(u_i) - \epsilon$
$E(H) := (E(H) \cup \{u_i w_i\}) \setminus \{v w_i\}$
$w(v) := 1$

Comment Note that $w_1, \ldots, w_t$ are distinct vertices, while $u_1, \ldots, u_t$ need not be. Note that $\{u_1, u_2, \ldots, u_t\} \cap \{w_1, \ldots, w_t\} = \emptyset$ since all w_i's are allocated, and thus not free, when the subroutine is called. If some u is the chosen free neighbor of several, say k, vertices from $w_1, \ldots, w_t$, then its weight drops by $k\epsilon$ and also k edges starting

in u are added to H. Lemma 5 shows that each w_i has indeed a free neighbor different from v in G.

Subroutine BranchA$(v, S, \overline{S}, w, H, B)$
1. $S' := S$; $\overline{S'} := \overline{S} \cup \{v\}$; $w' := w$; $w'(v) := 0$; $H' := H$
   SigmaRhoA$(S', \overline{S'}, w', H', B)$
2. let u be a free neighbor of v
   $S := S \cup \{v\}$; $w(v) := 0$; $w(u) := w(u) - \epsilon$; $E(H) := E(H) \cup \{uv\}$
   SigmaRhoA$(S, \overline{S}, w, H, B)$

Having described the recursive procedure and its subroutines, the entire algorithm named **Main-EnumSigmaRho-A** (see above) can be formalized as one call of the recursive procedure (and necessary prepossessing and final check of the items in the candidate list). See Lemma 9 for a further discussion of crucial properties of the procedures and their interplay.

The correctness of the algorithm follows from the following lemmas, from the fact that it branches on each vertex whose membership in S or $\overline{S}$ is not forced, and since each set S of L is explicitly checked for being (σ, ϱ)-dominating.

Lemma 2 (Weights and charges)

(i) *An allocated vertex v has always weight $w(v) = 0$.*
(ii) *At the time of each call of the **SigmaRhoA** procedure, the weight of a free vertex x is $w(x) = 1 - d\epsilon$, where d is the outdegree of x in H.*
(iii) *Whenever any of the subroutines has been terminated, the oriented graph H is a disjoint union of out-oriented stars, and $xy \in E(H)$ implies that $y \in S$.*
(iv) *The weight of a free vertex x is always nonnegative*: $w(x) \geq 0$.

Proof (i) *Weight of allocated vertices.* The weight of a vertex allocated to S or $\overline{S}$ becomes 0 at the time of allocation and remains unchanged afterwards.

(ii) *The weight of free vertices.* At the beginning the weight of every vertex is 1, and also H is edgeless, so the outdegree of every vertex is 0 (in H). The invariant follows by induction on the number of recursive calls. The weights of free vertices are changed in the **RechargeA** and **BranchA** subroutines, and in each case the multiple of ϵ subtracted from (or added to) the weight of the vertex is the same as the number of oriented edges starting in the vertex that are added to (deleted from, respectively) H.

(iii) *The shape of H.* First it is worth noting that the charge sent along an edge of H is always of value ϵ. Clearly, at the beginning of algorithm A the oriented graph H is edgeless. It gets modified in the **RechargeA** and **BranchA** subroutines. When recharging, an edge vw_i may be replaced by an edge $u_i w_i$ in H. When branching on v and selecting v, the edge uv is added to H. Therefore, the endpoint of an edge in H is always a vertex allocated to S and every vertex of S is the end vertex of at most one edge of H.

On the other hand, the start point of an edge added to H is always a free vertex at the time when the edge is added to H. However **ForcingA** might later allocate such a free vertex while keeping the edge in H. Thus H may contain an edge with a start vertex $x \in S$ and an end vertex $y \in S$. However in this case x has been allocated to S by forcing, and there has never been a branching on x. Consequently no edge of H has end point x. Therefore, H is indeed a union of out-oriented stars.

(iv) *Weights of free vertices are nonnegative.* A free vertex, say x, would have weight less than 0 only if it had outdegree $t > \max\{p, q\}$ in H. But then x must have t neighbors in S. Certainly at the beginning of the first call of **SigmaRhoA**, no such vertex exists. The number of S-neighbors may get raised during the first part of the **ForcingA** subroutine, but that is immediately discovered by the second part of the subroutine and the execution is halted. The only other possibility is during the second part of the **BranchA** subroutine, when v is selected into S. If there is a free neighbor x of v such that x has outdegree $t > \max\{p, q\}$ in H, when calling **SigmaRhoA**, then the free vertex x had exactly $\max\{p, q\}$ neighbors in S at the previous call of the **ForcingA** subroutine. Hence v would have been placed into $\overline{S}$, and never been considered for branching. Consequently $w(x) < 0$ for a free vertex is impossible. □

Lemma 3 (Halting) *If **ForcingA** halted with current values $S, \overline{S}$, then G contains no (σ, ϱ)-dominating set M such that $S \subseteq M \subseteq V \setminus \overline{S}$.*

Proof If **ForcingA** halts because some x has more than $\max\{p, q\}$ neighbors in S, then such an S cannot be a subset of any (σ, ϱ)-dominating set M. Indeed, if $x \in M$ then $|N(x) \cap M| \geq |N(x) \cap S| > p = \max \sigma$ and $|N(x) \cap M|$ cannot be in σ, as well as $|N(x) \cap M| \geq |N(x) \cap S| > q = \max \varrho$ and $|N(x) \cap M|$ cannot be in ϱ if $x \notin M$. If **ForcingA** halts because some $x \in S$ has a unique free neighbor, but neither $|N(x) \cap S|$ nor $|N(x) \cap S| + 1$ are in σ, then no M containing S is a (σ, ϱ)-dominating set since $|N(x) \cap M|$ equals $|N(x) \cap S|$ or $|N(x) \cap S| + 1$, depending on whether $v \in M$ or not. □

Lemma 4 (Necessity) *If at some stage, with current values of $S, \overline{S}$, **ForcingA** wants to place x in S (resp. in $\overline{S}$), then for every (σ, ϱ)-dominating set in G such that $S \subseteq M \subseteq V \setminus \overline{S}$, it holds that $x \in M$ (resp. $x \notin M$).*

Proof Assume that M is a (σ, ϱ)-dominating set such that $S \subseteq M \subseteq V \setminus \overline{S}$. Suppose that v is adjacent to a free vertex x such, that $|N(x) \cap S| = \max\{p, q\}$. If $v \in M$ then $|N(x) \cap M| > \max\{p, q\}$ and this is impossible, so $v \in \overline{M}$. Suppose now that v is the unique neighbor of $x \in S$ and $|N(x) \cap S| \in \sigma$. Then $|N(x) \cap S| + 1 \notin \sigma$ because σ is successor-free. Thus v cannot be in M, since then $|N(x) \cap M| = |N(x) \cap S| + 1 \notin \sigma$. Similarly, $|N(x) \cap S| + 1 \in \sigma$ implies $|N(x) \cap S| \notin \sigma$ and v must be in M, since it is the only possible additional M-neighbor of x. □

Lemma 5 (Correctness) *The subroutines **RechargeA** and **BranchA** can always be executed.*

Proof The vertex v chosen in the **SigmaRhoA** subroutine is allocated either in **ForcingA** or the next executions of **BranchA**. If v is allocated in **ForcingA** then **SigmaRhoA** is called recursively to choose the next vertex to allocate.

The **ForcingA** subroutine guarantees that no S-neighbor of v has v as its only free neighbor. This is crucial for the algorithm since it guarantees that recharging is possible. In **RechargeA**, for all $i = 1, 2, \ldots, t$, vw_i is an edge of H and hence $w_i \in S$. But then each w_i has another free neighbor and **RechargeA** does not get stuck.

For the **BranchA** subroutine, we note that vertices of G get allocated into S or $\overline{S}$ only when we attempt to branch on them (in the preceding **ForcingA** subroutine, or in **BranchA** itself). Thus the vertices are allocated in the reverse BFS ordering. Therefore when v is the last free vertex in the BFS ordering of the vertex set of G, either $v = v_1$ is the root (and then we do not bother checking anything and just add both S and $S \cup \{v\}$ to the candidate list L) or v has a predecessor u in the BFS tree of G. This u comes earlier in the BFS ordering of G, hence was not attempted to branch on yet, and hence is free at the time when v is processed. □

Analysis of the Running Time The weight of an instance $(G, w, S, \overline{S}, H)$ is $w(G) = \sum_{v \in V} w(v)$. In each branching on a vertex v the measure of the input decreases by 1 when discarding v, and it decreases by $1 + \epsilon$ when selecting v. In the standard terminology of branching algorithms this implies that the branching vector is $(1, 1 + \epsilon)$. The running time of each execution of **SigmaRhoA** (without recursive calls) is polynomial, and so the total running time is $O^*(T)$ where T is the number of leaves of the search tree. Note that each (σ, ϱ)-dominating set corresponds to one leaf of the search tree.

Let $T[k]$ be the maximum number of leaves of the search tree that any execution of our algorithm may generate on a problem instance of weight k. Due to the branching vector we obtain:

$$T[k] \leq T[k-1] + T[k-1-\epsilon].$$

Thus the number of (σ, ϱ)-dominating sets (which is bounded from above by $T[n]$) in an isolate-free graph on n vertices is $O^*(c^n)$, and the running time of our algorithm that enumerates all of them is $O^*(c^n)$, where c is the largest real root of the characteristic polynomial

$$x^{1+\epsilon} - x^{\epsilon} - 1.$$

The table shows the base of the exponential function bounding the running time of our algorithm for some particular values of $\varphi = \max\{p, q\}$.

φ	1	2	3	4	5	6	7	8	9	100
c	1.6181	1.7549	1.8192	1.8567	1.8813	1.8987	1.9116	1.9216	1.9296	1.9932

As can be expected, c converges to 2 when φ converges to infinity (and ϵ converges to 0). This is easily seen from the characteristic polynomial, which converges to $x - 2$.

3.2 Algorithm B

Now we assume that σ is successor-free, at least one of the sets σ and ϱ is finite, and $\sigma \cap \varrho = \emptyset$ (case B). Recall that $p = \max \sigma$ and $q = \max \varrho$. Algorithm B demonstrates the potentials of our approach. Contrary to algorithm A of the previous subsection, the recharging and the branching in algorithm B are more sophisticated. Instead of ϵ being the unique value of a charge, various values of charge might be sent now (along an edge of H). Therefore we denote the charge of an edge $e \in E(H)$ by $a(e)$. Thus $a(xy)$ is the charge sent from vertex x to vertex y along the edge $xy \in E(H)$.

To discuss the major refinement in algorithm B compared to Algorithm A, let us consider the subroutine **BranchB**. When branching on a free vertex v its weight $w(v)$ is equal to 1 which is guaranteed by the subroutine **RechargeB**. In algorithm A there is one way of branching and one way to send a necessary charge of ϵ from a free neighbor. In algorithm B there are two ways of branching. Firstly either v is discarded and the weight of the instance is decreased by 1, or v is selected and the weight of the instance is decreased by $1 + \epsilon$ by sending v a charge ϵ_1 from one free neighbor (in G) and a charge ϵ_2 from another free neighbor (in G) such that $\epsilon = \epsilon_1 + \epsilon_2$. Hence the branching vector is $(1, 1 + \epsilon)$. Secondly, if v has only one free neighbor then the weight of v will be increased by a charge of ϵ_3 sent by its unique free neighbor, and this happens when v is selected and *also* when v is discarded. Hence the branching vector is $(1 + \epsilon_3, 1 + \epsilon_3)$.

Although the value of ϵ will be fixed later as to minimize the constant c (depending on p and q) and thus the upper bound on the running time, some constraints concerning the choice of ϵ and ϵ_i for $i = 1, 2, 3$ are crucial for the correctness proof and need to be mentioned here. We set $\delta = \frac{1}{1+\min\{p,q\}}$, a natural value in such a Branch & Recharge algorithm. Then ϵ is chosen such that it satisfies the inequality $\delta < \epsilon \leq \delta + \delta^2 = \frac{2+\min\{p,q\}}{(1+\min\{p,q\})^2}$. The values of ϵ_1 and ϵ_2 will depend on the vertex v on which the algorithm branches. The only conditions to be satisfied are $\epsilon_1(v) + \epsilon_2(v) = \epsilon$ and $0 < \epsilon_1(v), \epsilon_2(v) \leq \delta$ for all v; thus $\epsilon_i(v) \leq \delta$ for all $i = 1, 2$ and all v. Finally we set $\epsilon_3 = \delta - \min\{p, q\}(\epsilon - \delta)$. First of all this guarantees $\epsilon_3 > 0$ since $\epsilon - \delta \leq \delta^2$ and $\min\{p, q\}\delta^2 = \delta(\min\{p, q\}\frac{1}{1+\min\{p,q\}}) < \delta$. The reason for choosing the value of ϵ_3 in this way will become clear in the analysis (see Lemma 6(iv)).

We will now describe the details our Branch & Recharge algorithm B in pseudocode.

Algorithm Main-EnumSigmaRho-B(G)
Initialization: $L := \emptyset$; $S := \emptyset$; $\overline{S} := \emptyset$; $H := (V(G), \emptyset)$; forall $v \in V(G)$ do
$w(v) := 1$
SigmaRhoB($S, \overline{S}, w, H$)
Termination:
forall $S \in L$ do
  if S is not a (σ, ϱ)-dominating set in G then $L := L \setminus \{S\}$
output(L)

Procedure SigmaRhoB$(S, \overline{S}, w, H)$
if there is no free vertex then $L := L \cup \{S\}$
else
  let v be a free vertex with maximum number of free neighbors in G
  ForcingB$(S, \overline{S}, w, H)$
  if ForcingB halted then Halt
  if v is still free then
    RechargeB$(v, S, \overline{S}, w, H)$
    BranchB$(v, S, \overline{S}, w, H)$
  else SigmaRhoB$(S, \overline{S}, w, H)$

Subroutine ForcingB$(S, \overline{S}, w, H)$
while ($\exists x$ s.t. x is free and $|N(x) \cap S| > \min\{p, q\}$) or
($\exists y \in S$ with a unique free neighbor z in G) or ($\exists$ a free vertex u with no free neighbor in G) do
  let x or y, z or u be such vertices
  case
    $|N(x) \cap S| > \max\{p, q\}$ then Halt
    $|N(x) \cap S| > p$ then $\overline{S} := \overline{S} \cup \{x\}$; $w(x) := 0$
    $|N(x) \cap S| > q$ then $S := S \cup \{x\}$; $w(x) := 0$
    $|N(y) \cap S| \in \sigma$ then $\overline{S} := \overline{S} \cup \{z\}$; $w(z) := 0$
    $|N(y) \cap S| + 1 \in \sigma$ then $S := S \cup \{z\}$; $w(z) := 0$
    $\{|N(y) \cap S|, |N(y) \cap S| + 1\} \cap \sigma = \emptyset$ then Halt
    $|N(u) \cap S| \in \sigma$ then $S := S \cup \{u\}$; $w(u) := 0$
    $|N(u) \cap S| \in \varrho$ then $\overline{S} := \overline{S} \cup \{u\}$; $w(u) := 0$
    $|N(u) \cap S| \notin \sigma \cup \varrho$ then Halt

Subroutine RechargeB$(v, S, \overline{S}, w, H)$
if $w(v) < 1$ then
  let $\{z_1, \ldots, z_t\} = \{x : vx \in E(H)\}$
  for $i := 1$ to t do
    if $\exists$ free neighbor x of z_i (in G) s.t. $xz_i \notin E(H)$ then
      $w(x) := w(x) - a(vz_i)$
      $E(H) := E(H) \cup \{xz_i\}$
      $a(xz_i) := a(vz_i)$
    else
      let x be a free neighbor of z_i s.t. $xz_i \in E(H)$ and $x \neq v$
      $w(x) := w(x) - a(vz_i)$
      $a(xz_i) := a(xz_i) + a(vz_i)$
    $E(H) := E(H) \setminus \{vz_i\}$
  $w(v) := 1$

Subroutine BranchB$(v, S, \overline{S}, w, H)$
if v has two different free neighbors x and y then
1. $S' := S$; $\overline{S'} := \overline{S} \cup \{v\}$; $w' := w$; $w'(v) := 0$; $H' := H$;
SigmaRhoB$(S', \overline{S'}, w', H')$
2. $S := S \cup \{v\}$; $w(v) := 0$
$E(H) := E(H) \cup \{xv, yv\}$; $a(xv) := \min\{\delta, w(x)\}$; $a(yv) := \epsilon - a(xv)$
$w(x) := w(x) - a(xv)$; $w(y) := w(y) - a(yv)$; SigmaRhoB$(S, \overline{S}, w, H)$
else
let x be the unique free neighbor of v
$E(H) := E(H) \cup \{xv\}$; $a(xv) := \epsilon_3$; $w(x) := w(x) - \epsilon_3$; $w(v) = 0$
1. $S' := S$; $\overline{S'} := \overline{S} \cup \{v\}$; $w' := w$; $H' := H$; SigmaRhoB$(S', \overline{S'}, w', H')$
2. $S := S \cup \{v\}$; SigmaRhoB$(S, \overline{S}, w, H)$

Algorithm B is a classical branching algorithm using a so-called maximum degree rule, namely **BranchB** branches on a free vertex with maximum number of free neighbors. See Lemma 9 for a further discussion of crucial properties of the procedures and their interplay.

The correctness of algorithm B follows from the subsequent four technical lemmas.

Lemma 6 (Weights and charges)

(i) *An allocated vertex v has always weight* 0.
(ii) *At the time of a call of the **SigmaRhoB** procedure, every vertex of H has indegree at most two, and either indegree or outdegree zero. After the termination of any **ForcingB** subroutine, x free and $xy \in E(H)$ implies $y \in S$.*
(iii) *At the time of each call of the **SigmaRhoB** procedure, the weight of a free vertex x satisfies $w(x) \geq 1 - d\epsilon$, where d is the outdegree of x in H. Furthermore after the termination of any **ForcingB** subroutine, $w(x) \geq 1 - d\delta$ for each free vertex x.*
(iv) *The weight of a free vertex is always nonnegative. Furthermore, after the termination of any **ForcingB** subroutine, $w(x) \geq \delta$ for each free vertex x.*

Proof (i) *The weight of allocated vertices.* The weight of a vertex allocated to S or $\overline{S}$ becomes 0 at the time of allocation and remains unchanged afterwards.

(ii) *The shape of H.* At the beginning, H is edgeless. It is modified only in the **RechargeB** and **BranchB** subroutines. When recharging, either (in the first part) edge vz_i is replaced by edge xz_i, or (in the second part) edge vz_i is removed. When branching, in the first type of branching, when v is discarded then H remains unchanged, when v is selected then two edges xv and yv are added to H. In the second type of branching, an edge xv is added to H, no matter whether v is discarded or selected.

Clearly allocated vertices have out-degree 0. Furthermore, if $xy \in H$ then y is allocated. Thus free vertices have indegree zero in H. Thus, when calling **SigmaRhoB**

for a free vertex v, the indegree of v in H is zero. Therefore at the termination of **BranchB** the indegree of the now allocated vertex v is at most two. Only recharging may add an edge xy to H where y is allocated, however this never increases the indegree of such a vertex y. Consequently allocated vertices have indegree at most two in H.

We have seen that if $xy \in E(H)$ then y is allocated. Suppose that $y \in \overline{S}$. Such an edge is only added to H in case of a second type branching. Hence x was the unique free neighbor of y when y was allocated to $\overline{S}$ by branching and the edge xy was added to H. Since y was chosen as a free vertex with maximum number of free neighbors, the vertex x has no free neighbor after the allocation of y. Hence the next call of the **ForcingB** subroutine will select or discard x (or halt).

(iii) *The weight of free vertices.* As already discussed any charge $a(xy)$ assigned to an edge xy of H during **BranchB** has value at most δ. A higher charge can only be assigned to an edge during the second part of **RechargeB**: x is a free neighbor of z_i, $vz_i \in E(H)$ and $xz_i \in E(H)$. This implies $z_i \in S$, as shown above, and $a(vz_i) + a(xz_i) = \epsilon$ since only the first type of branching creates two charges sent to z_i. Consequently the charge assigned to xz_i during the recharging is ϵ. Thus any charge sent along an edge of H is at most ϵ. Consequently $w(x) = 1 - \sum_{xy \in E(H)} a(xy) \geq 1 - d\epsilon$.

As pointed out, $a(xy) > \delta$ only if $y = z_i$ and the **RechargeB** subroutine is called for the free vertex v adjacent to z_i. Hence after executing **RechargeB**, the vertex x is the unique free neighbor of z_i, and thus in the next call of **ForcingB** it will select or discard x (or halt). Hence $w(x) = 1 - \sum_{xy \in E(H)} a(xy) \geq 1 - d\delta$ for each free vertex x after the execution of **ForcingB**.

(iv) *Weights of free vertices are nonnegative.* Let x be a free vertex at the time of calling **SigmaRhoB**. Let us consider the execution of the previous **SigmaRhoB** that had been called for some vertex $v \neq x$. Note that v has been allocated during its execution. Let d be the outdegree of x in H and let $xz_i \in E(H)$ for $i = 1, 2, \ldots, d$. By (ii), all z_i's are allocated and even more, at most one might be allocated to $\overline{S}$ (by a second type branching in the previous call of **SigmaRhoB**; since the next call of **ForcingB** would allocate x or halt).

The vertex x had $t \leq \min\{p, q\}$ neighbors $w_1, w_2, \ldots, w_t$ in S when the previous call of **ForcingB** was terminated, otherwise the subroutine would have allocated x or halted. Let us consider this particular moment during the execution of algorithm B. By (iii), $w(x) \geq 1 - d\delta$, where d is the outdegree of x in H, and by (ii), the endpoint of an edge belongs to S if its starting point is a free vertex, and thus $d \leq t$. Furthermore, any charge sent along an edge of H is at most $\delta = \frac{1}{1+\min\{p,q\}}$. Hence $w(x) \geq 1 - d\delta \geq 1 - \frac{\min\{p,q\}}{1+\min\{p,q\}} = \frac{1}{1+\min\{p,q\}}$. Consequently $w(x) \geq \delta$.

Now we consider the **RechargeB** subroutine when called for v. Clearly $z_i \in S$. In the first part an edge xz_i is added and the former charge $a(vz_i) \leq \delta$ is assigned to it. Since $z_i \in S$, all other at most $t - 1$ S-neighbors of x are endpoints of an edge from x with charge at most δ (if any) and thus $w(x) \geq 1 - (\min\{p, q\} - 1)\delta = 2\delta > 0$. In the second part the edge vz_i is deleted and the charge of xz_i is increased to ϵ. Hence $w(x) \geq 1 - (\min\{p, q\} - 1)\delta - \epsilon \geq \delta - \delta^2 > 0$.

Finally we consider the **BranchB** subroutine when called for v. In the first type of branching there are two free vertices x and y and edges xv and yv are added

when selecting v. As we have shown, when calling **BranchB** for v, both free vertices x and y have weight at least $\delta - \delta^2$. The vertices x and y need to recharge v by a value of δ in total. Their common weight $w(x) + w(y) \geq 2\delta - 2\delta^2$ is greater than or equal to δ if $\delta \leq 1/3$; and this is the case since x has at least two S-neighbors. In the second type of branching an edge xv with charge ϵ_3 for a free vertex x is added. As shown above, $w(x) \geq 1 - (\min\{p, q\} - 1)\delta - \epsilon \geq 2\delta - \epsilon$. The following justifies our choice of ϵ_3. We need that $\epsilon_3 \leq 2\delta - \epsilon$ to guarantee $w(x) \geq 0$. Since $\delta < \epsilon$ we obtain $\delta(\min\{p, q\} - 1) < \epsilon(\min\{p, q\} - 1)$. Therefore $\epsilon_3 = \delta + \delta \min\{p, q\} - \epsilon \min\{p, q\} \leq 2\delta - \epsilon \leq w(x)$.

Summarizing for a free vertex x we have $w(x) \geq 0$ at any time of the execution of the algorithm. □

The next two lemmas show the correctness of **ForcingB**.

Lemma 7 (Halting) *If **ForcingB** halts with current values $S, \overline{S}$, then G contains no (σ, ϱ)-dominating set M such that $S \subseteq M \subseteq V \setminus \overline{S}$.*

Proof If **ForcingB** halts because some x has more than $\max\{p, q\}$ neighbors in S, then such an S cannot be a subset of any (σ, ϱ)-dominating set M. Indeed, if $x \in M$ then $|N(x) \cap M| \geq |N(x) \cap S| > p = \max \sigma$ and $|N(x) \cap M|$ cannot be in σ, as well as $|N(x) \cap M| \geq |N(x) \cap S| > q = \max \varrho$ and $|N(x) \cap M|$ cannot be in ϱ if $x \notin M$. If **ForcingB** halts because some $y \in S$ has a unique free neighbor z, but neither $|N(y) \cap S|$ nor $|N(y) \cap S| + 1$ are in σ, then no M containing S is a (σ, ϱ)-dominating set since $|N(y) \cap M|$ equals $|N(y) \cap S|$ or $|N(y) \cap S| + 1$, depending on whether $z \in M$ or not. If the subroutine halts because for some free vertex u without any free neighbor, $|N(u) \cap S|$ is neither in σ nor in ϱ, then no superset M of S can be a (σ, ϱ)-dominating set since u can be neither in M nor outside it. □

Lemma 8 (Necessity) *If at some stage, with current values of $S, \overline{S}$, **ForcingB** wants to place x in S (resp. in $\overline{S}$), then for every (σ, ϱ)-dominating set in G such that $S \subseteq M \subseteq V \setminus \overline{S}$, it holds that $x \in M$ (resp. $x \notin M$).*

Proof Assume that M is a (σ, ϱ)-dominating set such that $S \subseteq M \subseteq V \setminus \overline{S}$.

Suppose x is free and $|N(x) \cap S| > p$ (and $\leq q$ since the subroutine did not halt in the previous step), x cannot be in M because then $|N(x) \cap M| \geq |N(x) \cap S| > p = \max \sigma$ and $|N(x) \cap M|$ could not be in σ. If x free and $|N(x) \cap S| > q$, then x cannot be outside M because then $|N(x) \cap M| \geq |N(x) \cap S| > q = \max \varrho$ and $|N(x) \cap M|$ could not be in ϱ.

Suppose z is the unique free neighbor of $y \in S$. If $|N(y) \cap S| \in \sigma$ then $|N(y) \cap S| + 1 \notin \sigma$ because σ is successor-free. Thus z cannot be in M, since then $|N(y) \cap M| = |N(y) \cap S| + 1 \notin \sigma$. Similarly, if $|N(y) \cap S| + 1 \in \sigma$ then $|N(y) \cap S| \notin \sigma$ and z must be in M, since it is the only possible additional M-neighbor of $y \in S$.

Finally, suppose that u is a free vertex with no free neighbor. Then $|N(u) \cap M| = |N(u) \cap S|$ and the membership of u in M is uniquely determined since $\sigma \cap \varrho = \emptyset$. □

Lemma 9 (Correctness) *The subroutines **RechargeB** and **BranchB** can always be executed. Whenever algorithm B changes the weight of an instance* $(G, S, \overline{S}, w)$ *by **ForcingB** or **BranchB** then the weight of the instance* never *increases.*

Proof The **ForcingB** subroutine allocates free vertices without free neighbor. Furthermore it guarantees that no S-neighbor of a free vertex has only one free neighbor. This guarantees that in a possibly following **RechargeB** subroutine, for every $i = 1, 2, \ldots, t$, vw_i is an edge of H and hence $w_i \in S$. Consequently, each w_i has another free neighbor and **RechargeB** does not get stuck; recharging is possible.

Additionally, by Lemma 6(i) and (iv), the weight of any vertex during the execution of algorithm B is always nonnegative. Hence all charges to be transfered from one vertex to another one are always available. Since no vertex can have negative weight, the **ForcingB** subroutine may decrease the overall weight of the instance or keep it unchanged. Clearly, **RechargeB** will not change the weight of the instance (which is part of its underlying idea). □

Analysis of the Running Time The weight of an instance $(G, w, S, \overline{S}, H)$ is $w(G) = \sum_{v \in V} w(v)$. Recall that there are two types of branching. If we branch on a vertex v with at least two free neighbors then the weight of the input decreases by 1 when discarding v, and it decreases by $1 + \epsilon$ when selecting v, hence the branching vector is $(1, 1 + \epsilon)$. On the other hand, if we branch on a vertex with exactly one free neighbor then the weight of the input decreases by $1 + \epsilon_3 = 1 + \delta - \min\{p, q\}(\epsilon - \delta)$, when discarding v and also when selecting v. Hence the branching vector is $(1 + \epsilon_3, 1 + \epsilon_3)$. Therefore we obtain the following two recurrences to estimate the running time $O^*(T)$.

$$T[k] \leq T[k-1] + T[k-1-\epsilon],$$
$$T[k] \leq 2T[k-1-\epsilon_3].$$

To balance the two recurrences ϵ is chosen such that the characteristic polynomials

$$x^{1+\epsilon} - x^{\epsilon} - 1$$
$$x^{1+\epsilon_3} - 2$$

have the same unique positive real root c.

How to choose ϵ? First, if $\min\{p, q\} = 0$, then we choose $\epsilon = 2$. Now let us assume that $\min\{p, q\} \geq 1$. Hence, if c_1, c_2 are the unique positive real roots of the above mentioned two polynomials then $c_2 = 2^{\frac{1}{1+\epsilon_3}}$. Furthermore, $\epsilon = \delta$ implies $c_1 < c_2$, and $\epsilon = \delta + \delta^2$ implies $c_2 < c_1$. Consequently, it is possible to choose ϵ in such a way that $c = c_1 = c_2$, and this is our choice of ϵ.

ψ	0	1	2	3	4	5	6	7	8	100
c	1.4656	1.6957	1.7901	1.8393	1.8698	1.8905	1.9055	1.9169	1.9258	1.9932
ϵ	2	0.6873	0.4047	0.2875	0.2231	0.1823	0.1541	0.1335	0.1177	0.00995

The table shows the base of the exponential function bounding the running time of our algorithm and values of ϵ for some particular values of $\psi = \min\{p, q\}$.

4 Lower Bounds

This section discusses the combinatorial consequences of our Branch & Recharge algorithms. We have shown that (under certain assumptions on σ and ϱ) every isolate-free graph on n vertices contains at most $2^{n(1-\theta)}$ (σ, ϱ)-dominating sets, for some $\theta > 0$.

Taking $\sigma = \{0\}$ and $\varrho = \mathbb{N}$ we obtain the Independent Dominating Set problem and the (σ, ϱ)-dominating sets are precisely the maximal independent sets. Hence our theorem implies that the maximum number of maximal independent sets is upper bounded by $O^*(1.4656^n)$; while the bound of Moon and Moser [22] is 1.4423^n and this is asymptotically tight.

While the upper bound of Moon and Moser is tight, others like the one by Fomin et al. for the maximum number of minimal dominating sets [9] might not be tight. Likewise, our upper bounds established by a general approach are unlikely to be tight for all particular values of (σ, ϱ). Thus it is natural to look after lower bounds.

Let σ be the set of all even integers from the interval $[0, r-1]$, and ϱ be the set of all odd integers from this interval, where $r \geq 2$ is a positive integer. Consider $G = sK_r$, the disjoint union of s copies of the complete graph K_r. Clearly, this graph G has $2^{(r-1)s} = 2^{\frac{r-1}{r}n}$ (σ, ϱ)-dominating sets.

Since both σ and ϱ are finite, and $\sigma \cap \varrho = \emptyset$, our algorithms of both cases can be applied. The next table compares the bases of the exponential upper bounds given by our algorithms, distinguished as c_A and c_B, respectively, with the base $c_L = 2^{\frac{r-1}{r}}$ of the exponential function given by the lower bound from the above example.

r	2	3	4	5	6	7	8	9	101
c_A	1.6181	1.7549	1.8192	1.8567	1.8813	1.8987	1.9116	1.9216	1.9932
c_B	1.4656	1.6957	1.7901	1.8393	1.8698	1.8905	1.9055	1.9169	1.9932
c_L	1.4142	1.5874	1.6817	1.7411	1.7817	1.8114	1.8340	1.8517	1.9863

5 Open Problems

It would be interesting to investigate the complexity of enumerating all (σ, ϱ)-dominating sets for other sets σ and ϱ, as well as the relationship to combinatorial upper bounds on their numbers.

We conclude with the following concrete open problems.

Problem 1 *Characterize all pairs σ, ϱ for which every isolate-free n-vertex graph has at most $2^{\delta n}$ (σ, ϱ)-dominating sets for some $\delta < 1$.*

Is it possible to solve the counting problem #-(σ, ϱ)-DS in time $O^*(2^{\delta n})$ for some $\delta < 1$, in the following cases:

Problem 2 *The set σ is finite and the set ϱ is equal to $\mathbb{N}_0$.*

Problem 3 *Both sets σ and ϱ are complements of finite sets.*

Note that Problem 2 includes Independent Set, Induced Matching, Regular Subgraph and Bounded Degree Subgraph, while Problem 3 includes Dominating Set and Total Dominating Set. Thus in both cases the enumeration problem ENUM-(σ, ϱ)-DS cannot be solved in time faster than $\Theta(2^n)$.

The recharging mechanism of our algorithms strictly needs the successor-freeness of σ. Nevertheless it is an interesting question whether the successor-freeness of σ is a condition that can be avoided.

Acknowledgements We are grateful to the anonymous referees for their suggestions helping us to improve the presentation of the paper.

References

1. Beigel, R., Eppstein, D.: 3-coloring in time O(1.3289^n). J. Algorithms **54**, 168–204 (2005)
2. Björklund, A., Husfeldt, T.: Inclusion-exclusion algorithms for counting set partitions. In: Proceedings of FOCS 2006, pp. 575–582. IEEE Press, New York (2006)
3. Byskov, J.M.: Enumerating maximal independent sets with applications to graph colouring. Oper. Res. Lett. **32**, 547–556 (2004)
4. Byskov, J.M., Madsen, B.A., Skjernaa, B.: On the number of maximal bipartite subgraphs of a graph. J. Graph Theory **48**, 127–132 (2005)
5. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. J. Graph Algorithm Appl. **7**, 131–140 (2003)
6. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: domination—a case study. In: Proceedings of ICALP 2005. LNCS, vol. 3380, pp. 192–203. Springer, Berlin (2005)
7. Fomin, F.V., Golovach, P., Kratsch, D., Kratochvil, J., Liedloff, M.: Branch and recharge: exact algorithms for generalized domination. In: Proceedings of WADS 2007. LNCS, vol. 4619, pp. 508–519. Springer, Berlin (2007)
8. Fomin, F.V., Gaspers, S., Pyatkin, A.V., Razgon, I.: On the minimum feedback vertex set problem: exact and enumeration algorithms. Algorithmica **52**, 293–307 (2008)
9. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Combinatorial bounds via measure and conquer: bounding minimal dominating sets and applications. ACM Trans. Algorithms **5**(1), 9 (2008)
10. Fomin, F.V., Golovach, P.A., Kratochvil, J., Kratsch, D., Liedloff, M.: Sort and search: exact algorithms for generalized domination. Inf. Process. Lett. **109**, 795–798 (2009)
11. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM **56**(5), 25 (2009)
12. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: A simple $O(2^{0.288n})$ independent set algorithm. In: Proceedings of SODA 2006, pp. 18–25. SIAM, Philadelphia (2006)
13. Golovach, P., Kratochvíl, J.: Computational complexity of generalized domination: a complete dichotomy for chordal graphs. In: Proceedings of WG 2007. LNCS, vol. 4769, pp. 1–11. Springer, Berlin (2007)
14. Golovach, P., Kratochvíl, J.: Generalized domination in degenerate graphs: a complete dichotomy of computational complexity. In: Proceedings of TAMC 2008. LNCS, vol. 4978, pp. 182–191. Springer, Berlin (2008)
15. Golovach, P., Kratochvíl, J., Suchý, O.: Parameterized complexity of generalized domination problems. In: Proceedings of WG 2009. LNCS, vol. 5911, pp. 133–142. Springer, Berlin (2009)

16. Gupta, S., Raman, V., Saurabh, S.: Fast exponential algorithms for Maximum r-regular induced subgraph problems. In: Proceedings of FSTTCS 2006. LNCS, vol. 4337, pp. 139–151. Springer, Berlin (2006)
17. Halldorsson, M.M., Kratochvíl, J., Telle, J.A.: Independent sets with domination constraints. Discrete Appl. Math. **99**, 39–54 (2000)
18. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Fundamentals of Domination in Graphs. Dekker, New York (1998)
19. Heggernes, P., Telle, J.A.: Partitioning graphs into generalized dominating sets. Nord. J. Comput. **5**, 128–142 (1998)
20. Kullmann, O.: New methods for 3-SAT decision and worst-case analysis. Theor. Comput. Sci. **223**, 1–72 (1999)
21. Lawler, E.L.: A note on the complexity of the chromatic number problem. Inf. Process. Lett. **5**, 66–67 (1976)
22. Moon, J.W., Moser, L.: On cliques in graphs. Isr. J. Math. **5**, 23–28 (1965)
23. Rosen, K.H.: Discrete Mathematics and Its Applications. McGraw-Hill, New York (2007)
24. Telle, J.A.: Complexity of domination-type problems in graphs. Nord. J. Comput. **1**, 157–171 (1994)
25. Woeginger, G.J.: Exact algorithms for NP-hard problems: A survey. In: Combinatorial Optimization—Eureka, You Shrink! LNCS, vol. 2570, pp. 185–207. Springer, Berlin (2003)