Roman Census: Enumerating and Counting Roman Dominating Functions on Graph Classes

Faisal N. Abu-Khzam
Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

Henning Fernau
Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Kevin Mann
Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Abstract

The concept of Roman domination has recently been studied concerning enumerating and counting in F. N. Abu-Khzam et al. (WG 2022). More technically speaking, a function that assigns $0, 1, 2$ to the vertices of an undirected graph is called a *Roman dominating function* if each vertex assigned zero has a neighbor assigned two. Such a function is called *minimal* if decreasing any assignment to any vertex would yield a function that is no longer a Roman dominating function. It has been shown that minimal Roman dominating functions can be enumerated with polynomial delay, i.e., between any two outputs of a solution, no more than polynomial time will elapse. This contrasts what is known about minimal dominating sets, where the question whether or not these can be enumerated with polynomial delay is open for more than 40 years. This makes the concept of Roman domination rather special and interesting among the many variants of domination problems studied in the literature, as it has been shown for several of these variants that the question of enumerating minimal solutions is tightly linked to that of enumerating minimal dominating sets, see M. Kanté et al. in SIAM J. Disc. Math., 2014. The running time of the mentioned enumeration algorithm for minimal Roman dominating functions (Abu-Khzam et al., WG 2022) could be estimated as $\mathcal{O}(1.9332^n)$ on general graphs of order n. Here, we focus on special graph classes, as has been also done for enumerating minimal dominating sets before. More specifically, for chordal graphs, we present an enumeration algorithm running in time $\mathcal{O}(1.8940^n)$. It is unknown if this gives a tight bound on the maximum number of minimal Roman dominating functions in chordal graphs. For interval graphs, we can lower this time bound further to $\mathcal{O}(1.7321^n)$, which also matches the known lower bound concerning the maximum number of minimal Roman dominating functions. We can also provide a matching lower and upper bound for forests, which is (incidentally) the same, namely $\mathcal{O}^*(\sqrt{3}^n)$. Furthermore, we present an optimal enumeration algorithm running in time $\mathcal{O}^*(\sqrt[3]{3}^n)$ for split graphs and for cobipartite graphs, i.e., we can also give a matching lower bound example for these graph classes. Hence, our enumeration algorithms for interval graphs, forests, split graphs and cobipartite graphs are all optimal. The importance of our results stems from the fact that, for other types of domination problems, optimal enumeration algorithms are not always found.

Interestingly, we use a different form of analysis for the running times of our different algorithms, and the branchings had to be tailored and tweaked to obtain the intended optimality results. Our Roman dominating functions enumeration algorithm for trees and forests is distinctively different from the one for minimal dominating sets by Rote (SODA 2019).Our approach also allows to give concrete formulas for counting minimal Roman dominating functions on more concrete graph families like paths.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases special graph classes, counting problems, enumeration problems, domination problems, Roman domination

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.6

Related Version *Full Version*: https://arxiv.org/abs/2208.05261

1 Introduction

Roman Domination comes with a nice (hi)story, on how to position armies on the various regions to secure the Roman Empire with the smallest cost, measured in the number of armies. "To secure" means that either (1) a region r has at least one army or (2) a region r' neighboring r contains two armies, so that it can afford sending one army to the region r without diminishing r''s self-defense capabilities.

It is easy to view Roman Domination as a graph-theoretic problem, where the map is modeled as a graph. Roman Domination has received notable attention in the last two decades [7, 17, 23, 26, 40, 41, 44, 48, 49, 51]. Relevant to our work is the development of exact algorithms: Roman Domination can be solved in $\mathcal{O}(1.5014^n)$ time (and space), see [40,52,54]. More combinatorial studies can be found in [16,18,25,35,39,42,43,47,55,56,57] as well as in the more recent chapter on Roman domination of [34]. Although independently introduced in [46], the *differential* of a graph is tightly related, see also [1, 8, 9, 10]. To briefly summarize all these findings, in many ways concerning complexity, Roman Domination and Dominating Set behave exactly the same. There are two notable and related exceptions, as delineated in [2], concerning *extension problems* and *output-sensitive enumeration*.

Extension problems often arise from search-tree algorithms for their optimization counterpart as follows. Assume that a search-tree node corresponds to a partial solution (or pre-solution) U and instead of proceeding with the search-tree algorithm (by exploring all the possible paths from this node onward) we ask whether we can extend U to a meaningful solution S. In the case of Dominating Set, this means that S is an inclusion-wise minimal dominating set that contains U. Unfortunately, this Extension Dominating Set problem and many similar problems are NP-hard, see [6, 12, 14, 15, 37, 38, 45]. Even worse: when parameterized by the "pre-solution size," Extension Dominating Set is one of the few problems known to be complete for the parameterized complexity class W[3], as shown in [11]. This blocks any progress on the Hitting Set Transversal Problem by using extension test algorithms, which is the question whether all minimal hitting sets of a hypergraph can be enumerated with polynomial delay (or even output-polynomial) only. This question is open for four decades by now and is equivalent to several enumeration problems in logic, database theory and also to enumerating minimal dominating sets in graphs, see [22, 24, 29, 36].

By way of contrast and quite surprisingly, with an appropriate definition of the notion of minimality, the extension variant of Roman Domination is solvable in polynomial time [3]. This was the key observation to show that enumerating all minimal Roman dominating functions is possible with polynomial delay. This triggered further interest in looking into enumerating minimal Roman dominating functions on graph classes, as also done in the case of Dominating Set, see [5, 20, 21, 30, 32, 33]. The basis of the output-sensitive enumeration result of [2] was several combinatorial observations. Here, we find ways how to use the underlying combinatorial ideas for non-trivial enumeration algorithms for minimal Roman dominating functions in split graphs, cobipartite graphs, interval graphs, forests and chordal graphs and for counting these exactly for paths. All these graph classes will be explained in separate sections below. These exploits constitute the main results of this paper. More details can be found at the end of the next section. Due to lack of space, further technical details can be found in [4]. We summarize known bases of lower and upper bounds on the number of minimal (Roman) dominating sets (resp. functions) in the next table; new results are shown with boxes; for matching bounds, only one number is displayed; c.f. [2, 21, 27, 33]. Polynomial delay is achievable for the mentioned special graph classes for enumerating minimal dominating sets [36, 38], but it is unclear how to combine these

approaches with good input-sensitive enumeration, while all input-sensitive results concerning minimal Roman dominating functions can also be implemented with polynomial delay, by interleaving extension tests with branching.

graph class:	general	chordal	split	interval	forests	cobipartite
domination	1.5704 / 1.7159	$\sqrt[3]{3}$ / 1.5048	$\sqrt[3]{3}$	$\sqrt[3]{3}$	$\sqrt[13]{95}$	1.3195 / 1.3674
Roman dom.	1.7441 / 1.9332	$\boxed{\sqrt{3} / 1.8940}$	$\boxed{\sqrt[3]{3}}$	$\boxed{\sqrt{3}}$	$\boxed{\sqrt{3}}$	$\boxed{\sqrt[3]{3}}$

2 Definitions and Known Results

Let $\mathbb{N} = \{1, 2, 3, \dots\}$ be the set of positive integers. For $n \in \mathbb{N}$, let $[n] = \{m \in \mathbb{N} \mid m \leq n\}$. We only consider undirected simple graphs. Let $G = (V, E)$ be a graph. For $U \subseteq V$, $G[U]$ denotes the graph induced by U. For $v \in V$, $N_G(v) := \{u \in V \mid \{u, v\} \in E\}$ denotes the *open neighborhood* of v, while $N_G[v] := N_G(v) \cup \{v\}$ is the *closed neighborhood* of v. $|N_G(v)|$ is called the *degree* of v; a vertex of degree 1 is known as a *leaf*. We extend such set-valued functions $X : V \to 2^V$ to $X : 2^V \to 2^V$ by setting $X(U) = \bigcup_{u \in U} X(u)$. Subset $D \subseteq V$ is a *dominating set*, or ds for short, if $N_G[D] = V$. For $D \subseteq V$ and $v \in D$, define the *private neighborhood* of $v \in V$ with respect to D as $P_{G,D}(v) := N_G[v] \setminus N_G[D \setminus \{v\}]$. A function $f : V \to \{0, 1, 2\}$ is called a *Roman dominating function*, or rdf for short, if for each $v \in V$ with $f(v) = 0$, there exists a $u \in N_G(v)$ with $f(u) = 2$. Simplifying notation, we set $V_i(f) := \{v \in V \mid f(v) = i\}$ for $i \in \{0, 1, 2\}$. The *weight* w_f of a function $f : V \to \{0, 1, 2\}$ equals $|V_1| + 2|V_2|$. The Roman Domination problem asks, given G and an integer k, if there exists an rdf of weight at most k. Connecting to the original motivation, G models a map of regions, and if the region vertex v belongs to V_i, then we place i armies on v.

For defining the problem Extension Roman Domination, we first need to define the order $\leq$ on $\{0, 1, 2\}^V$: for $f, g \in \{0, 1, 2\}^V$, let $f \leq g$ if and only if $f(v) \leq g(v)$ for all $v \in V$. Thus, we extend the usual linear ordering $\leq$ on $\{0, 1, 2\}$ to functions mapping to $\{0, 1, 2\}$ in a pointwise manner. We call a function $f \in \{0, 1, 2\}^V$ a *minimal Roman dominating function* if and only if f is an rdf and there exists no rdf g, $g \neq f$, with $g \leq f$. The weights of minimal rdfs can vary considerably. Consider for example a star $K_{1,n}$ with center c. Then, $f_1(c) = 2$, $f_1(v) = 0$ otherwise; $f_2(v) = 1$ for all vertices v; $f_3(c) = 0$, $f_3(u) = 2$ for one $u \neq c$, $f_3(v) = 1$ otherwise, define three minimal rdfs with weights $w_{f_1} = 2$, and $w_{f_2} = w_{f_3} = n + 1$.

In [2], several combinatorial properties of minimal Roman dominating functions were derived that were central for obtaining a general algorithmic enumeration result and that are also important when studying special graph classes. This is summarized as follows.

▶ **Theorem 2.1.** *Let $G = (V, E)$ be a graph, $f : V \to \{0, 1, 2\}$ and abbreviate $G' := G[V_0(f) \cup V_2(f)]$. Then, f is a minimal rdf if and only if the following conditions hold:*

1. $N_G[V_2(f)] \cap V_1(f) = \emptyset$,
2. $\forall v \in V_2(f) : P_{G', V_2(f)}(v) \not\subseteq \{v\}$, *also called* privacy condition, *and*
3. $V_2(f)$ *is a minimal dominating set of G'.*

This combinatorial result has been the key to show a polynomial-time decision procedure for the extension problem (Given a graph $G = (V, E)$, a function $f : V \to \{0, 1, 2\}$, the question is if there is minimal rdf g with $f \leq g$). It can also be used to design enumeration algorithms that are input-sensitive. The simplest exploit is to branch on all vertices whether or not a vertex should belong to $V_2(f)$. Once $V_2(f)$ is fixed, its neighborhood will form $V_0(f)$ and the remaining vertices will be $V_1(f)$. For better running times, this approach has to be refined, see Section 5. We obtain estimates of running times for branching algorithms as explained in [28], including an introduction into the Measure-and-Conquer analysis.

3 Enumerating Minimal RDFs in Split and in Cobipartite Graphs

A split graph $G = (V, E)$ consists of a bipartition of V as C and I, such that C forms a clique and I is an independent set. Let $f : V \to \{0, 1, 2\}$ be a minimal rdf of G. If $V_2(f)$ contains both a vertex v_c from C and a vertex v_i from I, then v_i cannot find a private neighbor in G, contradicting the minimality of f. We can hence first branch to decide if $V_2(f) \subseteq C$ or if $V_2(f) \subseteq I$. After dealing with the simple case that $|V_2(f) \cap C| = 1$ separately, we can assume that all private neighbors of $V_2(f) \subseteq C$ are in I and that all private neighbors of $V_2(f) \subseteq I$ are in C. We will describe a simple branching algorithm in which we can assume to immediately delete vertices that are assigned the value 0, as they will be always dominated.

Case 1. One element of C is assigned a value of 2. We can guess this element in $\mathcal{O}(n)$ and proceed as follows.

1. Elements of C with no neighbors in I are assigned a value of zero.
2. Pick $v \in C$ with at least two neighbors in I and branch by either setting $f(v) = 2$ and assign 0 to vertices in $N(v) \cap I$ or $f(v) = 0$ (this leads to the branching vector $(3, 1)$).
3. When all elements of C have exactly one neighbor in I, pick some $v \in C$ with $N(v) \cap I = \{w\}$. Distinguish two cases.
   - 3.1 w has at least one other neighbor $x \in C$. Then either $f(v) = 2$, $f(w) = f(x) = 0$ (in fact, all neighbors of w are assigned 0), or $f(v) = 0$ (this leads to a $(3, 1)$ branch).
   - 3.2 $N(w) = \{v\}$: either $f(v) = 2$, $f(w) = 0$ or $f(v) = 0$, $f(w) = 1$ (this leads to the branching vector $(2, 2)$).

Case 2. No element of C is assigned a value of 2.

1. Then any isolated element of I is automatically assigned a value of 1 and can be deleted. Moreover, any element of C with no neighbors in I is assigned a value of 1 and deleted.
2. Pick a vertex v of degree at least two in I and branch by either setting $f(v) = 2$ and assigning 0 to all its neighbors or set $f(v) = 1$ (this leads to the branching vector $(3, 1)$).
3. When all elements of I are leaves, pick $v \in I$ with $N(v) \cap C = \{w\}$. Distinguish 2 cases.
   - 3.1 w has at least one more neighbor $x \in I$: either $f(v) = 2, f(w) = 0, f(x) = 1$ or $f(v) = 1$ (delete v) (this leads to the branching vector $(3, 1)$).
   - 3.2 $N(w) \cap I = \{v\}$: either $f(v) = 2, f(w) = 0$ or $f(v) = f(w) = 1$ (this leads to the branching vector $(2, 2)$).

Notice that the analysis of the recursion is very simple: an rdf f is gradually defined, and the branching vectors describe the number of newly defined vertices. The worst-case branching vector is $(1, 3)$, which leads to the following claim.

▶ **Proposition 3.1.** *All minimal rdfs in a split graph of order n are enumerable in $\mathcal{O}^*(1.4656^n)$.*

▶ Remark 3.2. For cobipartite graphs, a similar reasoning applies. Now, it could be possible that one vertex x of the bipartition side X finds its private neighbor p_x in X itself and that one vertex y of the other bipartition side Y finds its private neighbor p_y in Y, such that the edges xp_y and yp_x do not exist. If G contains no universal vertices, then irrespectively whether the $V_2(f)$-vertices lie only in X or in Y, there must be at least one other vertex in $V_2(f)$ on the same side. But this means that they must find their private neighbors on the other side. The branching is hence analogous to the split graph case.

$$C_{P,2,n} = \begin{cases} n-1 & \text{if } n \in [2] \\ C_{P,\overline{2},n-2}, & \text{if } n > 2 \end{cases} \quad C_{P,\overline{2},n} = \begin{cases} n & \text{if } n \in [3] \\ C_{P,\overline{2},n-1} + C_{P,2,n-2} + C_{P,2,n-3} + C_{P,\overline{2},n-3}, & \text{if } n > 3 \end{cases}$$

Figure 1 The mutual recurrences for determining the number of minimal rdfs on a path P_n.

The previous arguments are invalid in the case of bipartite graphs. Here, we conjecture that the general case is not really easier than the bipartite case, as with minimal ds enumeration.

However, we can boost our algorithm and its simple analysis to actually prove an optimal enumeration result. In order to do this, a rather straightforward refinement of the previous case analysis suffices. Together with Remark 3.2 as well as the trick of interleaving the branching with extension tests as described in [2], this refined branching algorithm proves:

▶ **Theorem 3.3.** *All minimal rdfs in a split graph or a cobipartite graph of order n can be enumerated in time $\mathcal{O}^*(\sqrt[3]{3}^n)$, using polynomial space and polynomial delay only.*

We can complement Theorem 3.3 by showing lower bound examples in the following that prove that our simple branching algorithm analysis is optimal for split and cobipartite graphs.

▶ **Theorem 3.4.** *There exist split and cobipartite graphs of order n with $\Omega(\sqrt[3]{3}^n)$ many minimal rdfs.*

Proof. We consider the graph $G_t = (C_t \cup I_t, E_t)$ with $C_t = \{c_1, \ldots, c_{2t}\}$, $I_t = \{v_1, \ldots, v_t\}$, $3t = n = |C_t \cup I_t|$ and $E_t = \binom{C_t}{2} \cup \{\{c_{2i-1}, v_i\}, \{c_{2i}, v_i\} \mid i \in [t]\}$ (for the cobipartite case, I_t is also a clique). Thus, $v_i \in I_t$ has degree 2. If $V_2(f) \subseteq C_t$, there are three ways to Roman-dominate any $v_i \in I_t$, $c_{2i-1}, c_{2i} \in C_t$ with a minimal rdf f: $f(c_{2i}) = 2$, $f(c_{2i-1}) = f(v_i) = 0$ or $f(c_{2i-1}) = 2$, $f(c_{2i}) = f(v_i) = 0$ or $f(v_i) = 1$, $f(c_{2i-1}) = f(c_{2i}) = 0$ (resp. $f(c_{2i-1}) = f(c_{2i}) = 1$, if $V_2(f) = \emptyset$). This yields $3^t = \sqrt[3]{3}^n$ many minimal rdfs. There can be at most $2^t = \sqrt[3]{2}^n$ minimal rdfs f on G_t with $V_2(f) \subseteq I_t$. Hence, G_t is a graph of order $n = 3t$ that has $\sqrt[3]{3}^n + \sqrt[3]{2}^n - 1 \in \Omega(\sqrt[3]{3}^n)$ many minimal rdfs. ◀

Minimal dominating sets in cobipartite graphs where all dominating set vertices belong to one clique only correspond to minimal rdfs with no vertex assigned 1. So, we can use our rdf enumeration algorithm to enumerate minimal dominating sets on cobipartite graphs. This improves on the hitherto best published algorithm from [19] but would be worse than [53].

4 Counting Minimal Roman Dominating Functions on Paths

The following is the main result of this section, devoted to counting.

▶ **Proposition 4.1.** *The number of minimal Roman dominating functions of a path P_n grows as $\mathcal{O}^*(c_{RD,P}^n)$, with $c_{RD,P} \leq 1.6852$.*

This should be compared with the recursion of Bród [13] that yields the following asymptotic behavior for the number of minimal dominating sets of a path with n vertices:

▶ **Corollary 4.2** (follows from [13])**.** *The number of minimal dominating sets of a path P_n grows as $\mathcal{O}^*(c_{D,P}^n)$, with $c_{D,P} \leq 1.4013$.*

As every minimal dominating set $D \subseteq V$ of a graph $G = (V, E)$ corresponds to the minimal rdf $f : V \to \{0, 1, 2\}$ with $V_2(f) = D$ and $V_0(f) = V \setminus D$, it is clear that $c_{\mathrm{D,P}} \leq c_{\mathrm{RD,P}}$.

Proof of Proposition 4.1. Let $C_{P,n}$ count the number of minimal rdfs of a P_n. Furthermore, let $C_{P,2,n}$ and $C_{P,\overline{2},n}$ denote the number of minimal rdfs of a P_n where the first vertex is assigned 2, or where it is decided that the first vertex is not assigned 2, respectively. Clearly, $C_{P,n} = C_{P,2,n} + C_{P,\overline{2},n}$. Consider $P_n = (V_n, E_n)$ with $V_n = \{v_i \mid i \in [n]\}$ and $E_n = \{v_i v_{i+1} \mid i \in [n-1]\}$. Let $n \geq 3$ and $f : V_n \to \{0, 1, 2\}$ be a minimal rdf.
If $f(v_1) = 2$, then $f(v_2) = 0$. Also $f(v_3) \neq 2$, as v_1 would not have a private neighbor but itself for $f(v_3) = 2$. This shows (including trivial initial cases) the left-hand side of Figure 1. If $f(v_1) \neq 2$, then we have two subcases: (a) if $f(v_1) = 1$, then we know $f(v_2) \neq 2$; (b) if $f(v_1) = 0$, then $f(v_2) = 2$ is enforced. But we know more compared to the initial situation: v_2 has already a private neighbor, namely v_1. Thus, we have further possibilities for v_3: $f(v_3) = 2$ or $f(v_3) = 0$. The first subcase is as before: v_3 has no private neighbor. If $f(v_3) = 0$, then either $f(v_4) = 2$ and v_4 has no private neighbor, or $f(v_4) \neq 2$; hence the recursions on the right-hand side of Figure 1. Keeping in mind that $C_{P,n-3} = C_{P,2,n-3} + C_{P,\overline{2},n-3}$, we see $C_{P,n} = C_{P,2,n} + C_{P,\overline{2},n} = C_{P,\overline{2},n-2} + C_{P,\overline{2},n-1} + C_{P,2,n-2} + C_{P,n-3} = C_{P,\overline{2},n-1} + C_{P,n-2} + C_{P,n-3}$. Conversely, $C_{P,n} = C_{P,2,n} + C_{P,\overline{2},n} = C_{P,\overline{2},n-2} + C_{P,\overline{2},n}$. Hence,

$$C_{P,n} = C_{P,\overline{2},n} + C_{P,\overline{2},n-2} = C_{P,\overline{2},n-1} + (C_{P,\overline{2},n-2} + C_{P,\overline{2},n-4}) + (C_{P,\overline{2},n-3} + C_{P,\overline{2},n-5}),$$

which gives, ignoring the cases for small values of n, the following single recursion:

$$C_{P,\overline{2},n} = C_{P,\overline{2},n-1} + C_{P,\overline{2},n-3} + C_{P,\overline{2},n-4} + C_{P,\overline{2},n-5} \approx 1.6852^n$$

As $C_{P,n} = C_{P,\overline{2},n-2} + C_{P,\overline{2},n}$, the same asymptotic behavior holds for $C_{P,n}$. ◀

We will further extend this result towards forests and towards interval graphs in the next sections, starting with a more general description of such branching algorithms.

5 A General Approach to Branching for Minimal RDFs

In this section, we sketch the general strategy that we apply for enumerating minimal rdfs. In most cases, the branching will look for a yet undecided vertex v (that we will call *active* henceforth) and will decide to label it with 2 in one branch and not to label it with 2 in the other branch. Now, in the first branch, we can say something about the neighbors of v as well: according to Theorem 2.1, they cannot be finally labelled with 1. We express this and similar properties by (always) splitting the vertex set V of the current graph $G = (V, E)$ into:

- A: active vertices. In the very beginning of the branching, all vertices are active.
- $\overline{V_i}$: vertices that cannot be assigned a value of i, $i \in \{1, 2\}$, due to previous decisions.
- V_0: set of vertices assigned a value of zero that are not yet dominated.

Sometimes, the branching also considers a vertex from $\overline{V_1}$, which will be assigned 0 (and hence is deleted) in the branch when it is not assigned 2. We can also call extendibility tests before doing the branching in order to achieve polynomial delay; see [2].

Possibly, we can also (temporarily) have (and speak of) vertex sets V_i (with $i \in \{1, 2\}$) with the meaning that each vertex in V_i is assigned the value i. Our algorithms will preserve the invariant that a vertex $v \in \overline{V_1}$ must have a neighbor put into V_2 (in the original graph), i.e., $N(v) \cap V_2 \neq \emptyset$, which is a property that can be exploited in our analysis. Namely, a vertex is put into $\overline{V_1}$ only if one of its neighbors has been put into V_2. However, notice that once the effect (mostly implied by Theorem 2.1) of putting a vertex v into V_i on its neighborhood $N(v)$ has been taken care of, such a vertex v can be deleted from the "current graph" to simplify the considerations. More precisely, for $i \in \{1, 2\}$, our algorithms automatically delete vertices assigned a value of i after making sure the neighbors are placed in $\overline{V_{3-i}}$. It could happen

that the neighbor of a vertex $w \in \overline{V_2}$ is assigned the value 2. Then, w must be assigned 0; as it is dominated, it can and will be deleted. Similarly, if the neighbor of a vertex $w \in \overline{V_1}$ is assigned the value 1, w must be assigned 0 and is hence deleted. Only finally, it should be checked if a function $f : V \to \{0, 1, 2\}$ that is constructed during branching is indeed a minimal rdf, as possibly some vertices assigned 2 do not have a private neighbor. During the course of our algorithm, whenever we speak of the degree of a vertex (in the current graph) in the following, we only count in neighbors in $A \cup \overline{V_1} \cup \overline{V_2}$. In most stages of our algorithms, we can assume $V_1 = \emptyset$, as we will explain.

Reduction rules are an important ingredient of any branching algorithm, as also shown in [28]. We will make use of the following reduction rules. Similar rules appeared in [2].

▶ Reduction Rule 5.1. If $v \in \overline{V_2}$ with $N(v) \subseteq \overline{V_2}$, then set $f(v) = 1$ and delete v.

▶ Reduction Rule 5.2. If $v \in \overline{V_1}$ with $N(v) \subseteq \overline{V_1}$, then set $f(v) = 0$ and delete v.

▶ Reduction Rule 5.3. If $v \in A$ with $N(v) \subseteq \overline{V_1}$, then put v into $\overline{V_2}$.

▶ **Lemma 5.1.** *The three presented reduction rules are sound.*

In contrast to our approach in Section 3, we will now perform a Measure-and-Conquer analysis of the branching algorithms that we will describe. As a measure, we take

$$\mu(A, \overline{V_1}, \overline{V_2}, V_0) = |A| + \omega_1 |\overline{V_1}| + \omega_2 |\overline{V_2}|$$

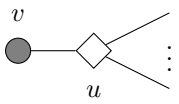
for the "current graph" with vertex set partitioned as $A \cup \overline{V_1} \cup \overline{V_2} \cup V_0$. Hence, whenever we measure our graph, we can assume $V_1 = \emptyset$. In the beginning of the algorithm, $A = V$ and $\overline{V_1} = \overline{V_2} = V_0 = V_1 = \emptyset$. To explain the work of the reduction rules, consider an isolated vertex (in the very beginning). The reduction rules will first move it into $\overline{V_2}$ and then into V_1 to finally delete it. We will choose the constants $\omega_1, \omega_2 \in [0, 1]$ to assess the running times of our algorithms best possible, hence also delivering upper bounds on the number of minimal rdfs of graphs of order n belonging to a specific graph class.

Concerning the reduction rules, we can easily observe that their application will never increase the measure. We will list in the following several branching rules (for the different graph classes) and we always assume that the rules are carried out in the given order.

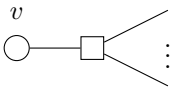
6 Enumerating Minimal RDFs on Interval Graphs and Forests

Recall that an *interval graph* can be described as the intersection graph of a collection of intervals on the real line. This means that the vertices correspond to intervals and that there is an edge between two such vertices if the intervals have a non-empty intersection. We assume in the following that $G = (V, E)$ is an interval graph with the interval representation $\mathcal{I} = \{I_v := [l_v, r_v]\}_{v \in V}$, i.e., l_v is the left border and r_v is the right border of the interval representing the vertex v. We call $v \in U$ *leftmost in* $U \subseteq V$ if it is a vertex from U that has the smallest value of r_u among all vertices in U. A vertex leftmost in V is simply called leftmost. Notice that this notion of a leftmost vertex will be used in many places in the rules exhibited in the following and is not available in the setting of general graphs as investigated in [3] but relies on the interval graph structure. Our algorithm always branches on the leftmost vertex. Then, it simply considers all cases. We now present more details.

The reduction rules from Section 5 imply that each vertex in $v \in A$ has at least one neighbor in $A \cup \overline{V_2}$. Concerning the measure, we will have $\omega_1 = 1$ and set $\omega_2 = \omega = 0.57$. We will present the branching rules that constitute the backbone of our algorithm for enumerating minimal rdfs on interval graphs. We often provide illustrations of the different branching scenarios. In our figures, we adhere to the following drawing conventions:

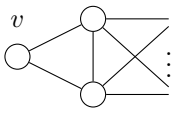


(a) v is already dominated.

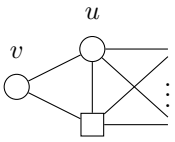


(b) $v \in A$ has neighbors in $\overline{V_2}$.

Figure 2 Branching Rules 6.1 and 6.2. Here (and elsewhere in these illustrations) we only sketch important parts of a subgraph, not necessarily covering all cases of the rules within the drawings.



(a) $v \in A$ has neighbors in A.



(b) $v \in A$ has neighbors in $\overline{V_2}$ and in A.

Figure 3 Branching Rules 6.3 and 6.4.

- ○ are vertices in A, ● are vertices in $\overline{V_1}$, ◒ are vertices in $A \cup \overline{V_1}$.
- □ are vertices in $\overline{V_2}$.
- ◇ are vertices in $A \cup \overline{V_2}$, for which the exact set is not further defined.
- ◈ are vertices in $A \cup \overline{V_1} \cup \overline{V_2}$, for which the exact set is not further defined.

▶ Branching Rule 6.1. Let v be the leftmost vertex in $\overline{V_1}$ and let u be the leftmost vertex in $N(v) \cap (A \cup \overline{V_2})$ and branch as follows: (1) Put v in V_0. (2) Put v in V_2 and u in V_0.

▶ **Lemma 6.1.** *The branching of Branching Rule 6.1 is a complete case distinction. Moreover, it leads at worst to the following branching vector:* $(1, 1 + \omega)$.

One can formulate and prove similar lemmas for the other branching rules that we present; see [4]. The branching vectors and branching numbers are summarized in Table 1.

▶ Branching Rule 6.2. Let v be the leftmost vertex in $(A \cup \overline{V_2})$. If $v \in A$ and $N(v) \cap A = \emptyset$ hold, branch as follows: (1) Put v in V_2 and $N(v) \cap \overline{V_2}$ in V_0. (2) Put v in V_1.

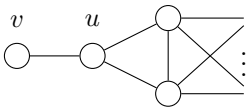
▶ Branching Rule 6.3. Let v be leftmost in $(A \cup \overline{V_2})$. If $v \in A$ and $|N(v) \cap A| \geq 2$ hold, branch as follows: (1) Put v in V_2 and all vertices in $N(v) \cap A$ into V_0. (2) Put v in $\overline{V_2}$.

▶ Branching Rule 6.4. Let v be the leftmost vertex in $(A \cup \overline{V_2})$. If $v \in A$, $|N(v) \cap \overline{V_2}| \geq 1$ and $|N(v) \cap A| = 1$ with $u \in N(v) \cap A$ hold, then branch: (1) Put v in V_2, $N(v) \cap (\{u\} \cup \overline{V_2})$ in V_0. (2) Put u in V_2 and $\{v\} \cup (N(v) \cap \overline{V_2})$ in V_0. (3) Put v in V_0 and u in $\overline{V_2}$.

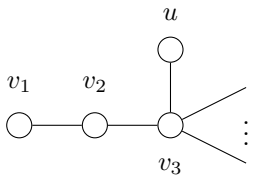
▶ Branching Rule 6.5. Let v be the leftmost vertex in $(A \cup \overline{V_2})$. If $N[v] \cap A = \{v, u\}$ with $N[v] \cap \overline{V_2} = \emptyset$ and $|N(u) \cap A| \geq 3$, branch as follows:
(1) Put v in V_2, u in V_0 and $N(u) \setminus \{v\}$ in $\overline{V_2}$. (2) Put v in $\overline{V_2}$.

▶ Branching Rule 6.6. Let v_1 be the leftmost vertex in $(A \cup \overline{V_2})$. If $N[v_1] \cap A = \{v_1, v_2\}$ with $N[v_1] \cap \overline{V_2} = \emptyset$, $N(v_2) \cap A = \{v_1, v_3\}$ and if there exists a $u \in N(v_3)$ such that $N(u) = \{v_3\}$, then branch as follows: (1) Put v_1 in V_2, v_2 in V_0 and v_3 in $\overline{V_2}$. (2) Put v_1 in V_1, v_2 in $\overline{V_2}$. (3) Put v_2 in V_2 and v_1, v_3 in V_0 and u in V_1. (4) Put v_2, v_3 in V_2 and v_1, u in V_0.

▶ Branching Rule 6.7. Let v_1 be the leftmost vertex in $(A \cup \overline{V_2})$, such that $N[v_1] \cap A = \{v_1, v_2\}$, with $N[v_2] \cap \overline{V_2} = \emptyset$ and $N(v_2) \cap (A \cup \overline{V_2}) = \{v_1, v_3\}$. If there is a u leftmost in $A \setminus \{v_1, v_2, v_3\}$, with $\{v_3\} \subsetneq N(u)$, then branch as follows: (1) Put v_1 in V_2 and v_2 in V_0 and v_3 in $\overline{V_2}$. (2) Put v_1 in V_1, v_2 in $\overline{V_2}$. (3) Put v_1 in V_0, v_2 in V_2 and v_3 in $\overline{V_2}$. (4) Put v_2, v_3 in V_2 and v_1, u in V_0 and $N(u) \setminus \{v_3\}$ in $\overline{V_2}$.

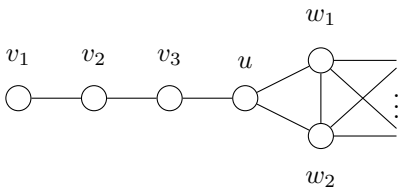


(a) $v_1 \in A$ has only on neighbor in A which has degree bigger than 2.

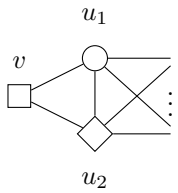


(b) $v_1, v_2, v_3 \in A$ is a path and v_3 has a leaf neighbor.

Figure 4 Branching Rules 6.5 and 6.6.



(a) $v_1, v_2, v_3 \in A$ is a path and there exists a $u \in A \cap N(v_3)$ with one more neighbor.



(b) $v \in \overline{V_2}$ is the leftmost vertex.

Figure 5 Rules 6.7 and 6.8.

▶ Branching Rule 6.8. Let v be the leftmost vertex in $(A \cup \overline{V_2})$. If $v \in \overline{V_2}$, branch like: (1) For each $u \in N(v) \cap A$: u in V_2 and $N[v] \setminus \{u\}$ into V_0. (2) Put v in V_1 and $N(v) \cap A$ in $\overline{V_2}$.

▶ **Theorem 6.2.** *All minimal rdfs of an interval graph of order n can be enumerated in time $\mathcal{O}^*\left(\sqrt{3}^n\right)$, with polynomial delay and in polynomial space.*

This result is optimal, as there are interval graphs that have $\sqrt{3}^n$ many minimal rdfs, namely collections of paths on two vertices: $x - y$ can be Roman-dominated by $f(x) = f(y) = 1$ or by assigning two to one vertex and zero to the other one, i.e., we get three possibilities per two vertices. For optimally enumerating minimal ds in interval graphs, see [31].
Recall that a *forest* is an acyclic undirected graph. A branching scenario that is similar to, but slightly more complex than, that of interval graphs can be used for forests (see [4]).

▶ **Theorem 6.3.** *A forest of order n has at most $\sqrt{3}^n$ many minimal rdfs. They can also be enumerated in time $\mathcal{O}^*\left(\sqrt{3}^n\right)$, with polynomial delay and in polynomial space.*

Table 1 Branching scenarios on interval graphs.

rule	branching vector	branching number
6.1 & 6.2	$(1, 1+\omega)$	1.7314
6.3	$(3, 1-\omega)$	1.6992
6.4	$(2+\omega, 2+\omega, 2-\omega)$	1.6829
6.5	$(4-2\omega, 1-\omega)$	1.7274
6.6	$(3-\omega, 2-\omega, 4, 4)$	1.6877
6.7	$(3-\omega, 2-\omega, 3, 5-\omega)$	1.7315
6.8	$(\underbrace{\omega+\lvert N(v)\cap A\rvert, \ldots, \omega+\lvert N(v)\cap A\rvert}_{\lvert N(v)\cap A\rvert \text{ many times}}, \omega+(1-\omega)\cdot\lvert N(v)\cap A\rvert)$	$\leq \sqrt{3} \leq 1.7321$

Table 2 Branching rules and their vectors and numbers for chordal graphs; worst cases in red.

Rule	branching vector	branching number
7.1	$(1-\omega_2, 1+3\min(1-\omega_1, \omega_2))$	1.8940
7.2	$(1+\omega_1+\omega_2, 1-\omega_2)$	1.8014
7.3	$(\omega_1, \omega_1+2\omega_2)$	1.8940
7.4	$(\omega_1, 2-\omega_1+\min(1-\omega_1, \omega_2))$	1.8940
7.5 & 7.16	$(\omega_1, 2\omega_1+\omega_2)$	1.7915
7.6	$(\omega_1+\omega_2, \omega_1+\omega_2)$	1.8321
7.7 & 7.14	$(1+\omega_2, 1+\omega_2)$	never worse than Branching Rule 7.6
7.8	$(1+\omega_2+\min(1-\omega_2, \omega_1), 1)$	1.6181
7.9	$(2, 1-\omega_2)$	1.8471
7.10	$(1+\omega_2, 1)$	1.779
7.11	$(1+\omega_1+2(1-\omega_2), \omega_1)$	1.5743
7.12	$(1+2\omega_2, 1)$	never worse than Branching Rule 7.10
7.13	$(2+\omega_2, 1-\omega_2)$	1.7249
7.15 & 7.17	$(2-\omega_1+\omega_2, 2+\omega_2, 2-\omega_2)$	1.8005

(a) $v \in A$ has at least 3 neighbors in $A \cup \overline{V_2}$.

(b) $v \in A$ has one neighbor $w \in \overline{V_2}$ and at least one neighbor in $\overline{V_1}$ that has only further neighbors in $\overline{V_1}$.

Figure 6 Branching Rules 7.1 and 7.2.

This result is again optimal, as there are forests that have $\sqrt{3}^n$ many minimal rdfs, namely collections of P_2. A similar optimality result was obtained by Rote [50] for enumerating minimal dominating sets in forests by using different techniques: there are (at most) $\sqrt[13]{95}^n$ many of them in forests of order n.

7 Enumerating Minimal RDFs in Chordal Graphs

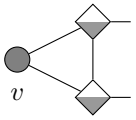
Recall that a graph is *chordal* if the only induced cycles it might contain have length three. In this quite technical section, we explain the following result whose optimality is open.

▶ **Theorem 7.1.** *All minimal Roman dominating functions of a chordal graph of order n can be enumerated with polynomial delay and in polynomial space in time $\mathcal{O}(1.8940^n)$.*

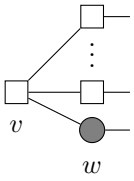
We are following the general approach sketched in Section 5. We adopt as a measure $\mu = |A| + \omega_1 |\overline{V_1}| + \omega_2 |\overline{V_2}|$. To obtain our result, we set $\omega_1 = 0.710134$ and $\omega_2 = 0.434799$.

Initially, all vertices are in A. Each branching rule assumes the preceding rules have been applied exhaustively and none of their conditions is applicable anymore. We omit stating correctness lemmas and lemmas concerning branching vectors but refer to Table 2. These lemmas are in general quite simple.

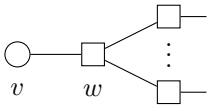
▶ Branching Rule 7.1. If $v \in A$ has at least three neighbors in $A \cup \overline{V_2}$, then we branch as follows: (1) Set $f(v) = 2$ and update the neighbors accordingly. (2) Add v to $\overline{V_2}$.



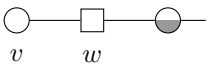
(a) $v \in \overline{V_2}$ is simplicial with at least 2 neighbors.



(b) $v \in \overline{V_2}$ has exact one neighbor in $\overline{V_1}$, possibly more neighbors in $\overline{V_2}$.



(c) $v \in A$ is a leaf with a neighbor in $\overline{V_2}$ which has only further neighbors in $\overline{V_2}$.



(d) $v \in A$ is a leaf with a neighbor $w \in \overline{V_2}$ with at least one neighbor in $A \cup \overline{V_1}$.

Figure 7 Branching Rules 7.5, 7.6, 7.7 and 7.8.

From now on, we can assume that a vertex from A of degree at least 3 has a neighbor in $\overline{V_1}$.

▶ Branching Rule 7.2. If $v \in A$ has at least one neighbor w in $\overline{V_2}$ and at least one neighbor u in $\overline{V_1}$ such that all neighbors of u (but v and possibly w) are in $\overline{V_1}$, then we branch as follows: (1) Set $f(v) = 2$ and update the neighbors accordingly. (2) Add v to $\overline{V_2}$.

Knowing (by our invariants) that elements of $\overline{V_1}$ are guaranteed to have neighbors in V_2, the next two branching rules apply to some elements of $\overline{V_1}$ (illustration can be found in [4]):

▶ Branching Rule 7.3. If $v \in \overline{V_1}$ has at least two neighbors in $\overline{V_2}$, then we branch as follows: (1) Set $f(v) = 2$ and update the neighbors accordingly. (2) Set $f(v) = 0$ and delete v.

▶ Branching Rule 7.4. If $v \in \overline{V_1}$ has at least three neighbors in $A \cup \overline{V_2}$ then we branch as follows: (1) Set $f(v) = 2$ and update the neighbors accordingly. (2) Set $f(v) = 0$ and delete v.

From now on, we discuss branching on simplicial vertices (or sometimes on vertices in the neighborhood of simplicial vertices as in [5]).

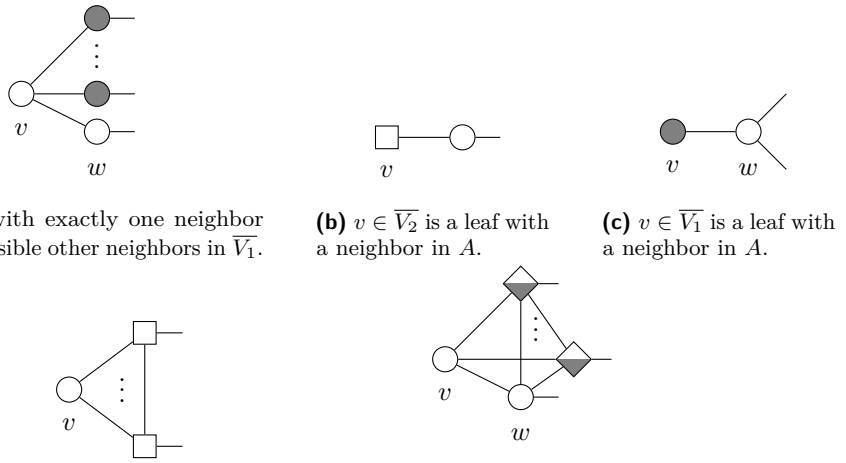
▶ **Observation 7.2.** *Simplicial vertices in $\overline{V_1}$ can only have neighbors in $A \cup \overline{V_2} \cup \overline{V_1}$. As we already considered vertices in $\overline{V_1}$ with ≥ 3 neighbors in $A \cup \overline{V_2}$, in the following branchings, a vertex in $\overline{V_1}$ has ≤ 2 neighbors in $A \cup \overline{V_2}$, not both of them in $\overline{V_2}$ due to Branching Rule 7.3.*

▶ Branching Rule 7.5. If $v \in \overline{V_1}$ is simplicial and of degree at least two, then branch as follows: (1) Set $f(v) = 2$ and update the neighbors accordingly. (2) Set $f(v) = 0$ and delete v.

▶ **Observation 7.3.** *We note that an isolated pair of adjacent leaves, say v, w, give rise to a path, which has already been studied. However, assuming previous branching rules have resulted in such a path, the worst case is when $v \in \overline{V_2}$ and $w \in \overline{V_1}$. To see this, note that if both v and w are in $\overline{V_2}$ or both in $\overline{V_1}$, they would be deleted by Reduction Rules 5.1 or 5.2.*

▶ Branching Rule 7.6. If $v \in \overline{V_2}$ is a vertex with exactly one neighbor $w \in \overline{V_1}$ and possibly more neighbors in $\overline{V_2}$, then we branch as follows: (1) Set $f(w) = 2$, $f(v) = 0$ and update the neighbors of w accordingly. (2) Set $f(w) = 0$ and $f(v) = 1$ and delete v, w.

▶ Branching Rule 7.7. Let $v \in A$ with $N(v) = \{w\}$, $w \in \overline{V_2}$, with $N(w) \setminus \{v\} \subseteq \overline{V_2}$. Then, branch as follows: (1) Set $f(v) = 2$ and $f(w) = 0$. (2) Set $f(v) = f(w) = 1$ and delete v, w.



(a) $v \in A$ with exactly one neighbor in A and possible other neighbors in $\overline{V_1}$.

(b) $v \in \overline{V_2}$ is a leaf with a neighbor in A.

(c) $v \in \overline{V_1}$ is a leaf with a neighbor in A.

(d) $v \in A$ is simplicial and has only neighbors in $\overline{V_2}$.

(e) $v \in A$ is a simplicial vertex of degree at least 3 and at least one neighbor in A.

Figure 8 Branching Rules 7.9, 7.10,7.11, 7.12 and 7.13.

▶ Branching Rule 7.8. If $v \in A$ with $N(v) = \{w\}$ and $w \in \overline{V_2}$ and if there is at least one further neighbor of w that belongs to $A \cup \overline{V_1}$, then we branch as follows: (1) Set $f(v) = 2$ and $f(w) = 0$ and update all neighbors of w to $\overline{V_2}$ or to V_0. (2) Set $f(v) = 1$ and delete v.

The following rule again deals with a leaf vertex as a special case.

▶ Branching Rule 7.9. Let $v \in A$ with $N(v) \cap A = \{w\}$ and $N(v) \setminus \{w\} \subseteq \overline{V_1}$. Then, branch: (1) Set $f(v) = 2$ and $f(w) = 0$, update all neighbors of w to $\overline{V_2}$ or to V_0. (2) Add v to $\overline{V_2}$.

▶ Branching Rule 7.10. If $v \in \overline{V_2}$ with $N(v) = \{w\}$, $w \in A$, then we branch as follows: (1) Set $f(w) = 2$ and $f(v) = 0$; update $N(w)$ accordingly. (2) Add w to $\overline{V_2}$ and set $f(v) = 1$.

▶ Branching Rule 7.11. If $v \in \overline{V_1}$ with $N(v) = \{w\}$, $w \in A$ and $|N(w) \cap A| = 2$, then branch: (1) Set $f(v) = 2$, $f(w) = 0$ and put the neighbors of w into $\overline{V_2}$. (2) Set $f(v) = 0$ and delete v.

▶ Branching Rule 7.12. If $v \in A$ is simplicial, of degree ≥ 2 with $N(v) \subset \overline{V_2}$, then branch: (1) Set $f(v) = 2$ and assign zero to all its neighbors (delete $N[v]$). (2) Set $f(v) = 1$ and delete v.

▶ Branching Rule 7.13. If $v \in A$ is simplicial, with $|N(v)| \geq 2$ and $N(v) \cap A \neq \emptyset$, then we branch as follows: (1) Set $f(v) = 2$ and update the neighbors accordingly. (2) Add v to $\overline{V_2}$.

Finally, we consider simplicial vertices in $\overline{V_2}$ of degree ≥ 2, now covering the remaining cases.

▶ Branching Rule 7.14. Let $v \in \overline{V_2}$ be a simplicial vertex of degree two with a neighbor $w \in A$. If the other neighbor w' of v is in $\overline{V_2}$, then we branch as follows: (1) Set $f(w) = 2$ and $f(v) = f(w') = 0$. (2) Add w to $\overline{V_2}$, set $f(v) = 1$ and delete v.

▶ Branching Rule 7.15. If $v \in \overline{V_2}$ is simplicial with two neighbors $w, w' \in A$, then we branch as follows: (1) Set $f(w) = 2$, $f(v) = 0$ and add w' to $\overline{V_1}$. (2) Set $f(w') = 2$ and $f(w) = f(v) = 0$. (3) Add w and w' to $\overline{V_2}$ and set $f(v) = 1$.

▶ Branching Rule 7.16. If $v \in \overline{V_2}$ is simplicial, of degree at least two, with a neighbor w such that $N[w] \setminus \{v\} \subseteq \overline{V_1}$, then we branch as follows: (1) Set $f(w) = 2$, $f(v) = 0$ and delete $N[v]$. (2) Set $f(w) = 0$ and delete it.

▶ Branching Rule 7.17. If $v \in \overline{V_2}$ is simplicial, with neighbors $w, w' \in \overline{V_1}$ s.t. $N[w] \subset N[w']$, then branch: (1) Set $f(w') = 2$, $f(v) = f(w) = 0$. (2) Set $f(w') = 0$ and delete it.

▶ **Lemma 7.4.** *Our rules cover all possible cases for chordal graphs.*

It remains open whether enumeration on chordal graphs can be improved further, so we hereby pose it as an open problem, or whether one can obtain a higher lower bound, which might also be a gap-improvement on general graphs. So far, the best lower bound for general graphs is a collection of C_5's [2], which is clearly not a chordal graph. The worst-case example for chordal graphs is a collection of P_2's, see Section 4 and our discussions on interval graphs.

References

1 F. N. Abu-Khzam, C. Bazgan, M. Chopin, and H. Fernau. Data reductions and combinatorial bounds for improved approximation algorithms. *Journal of Computer and System Sciences*, 82(3):503–520, 2016.

2 F. N. Abu-Khzam, H. Fernau, and K. Mann. Minimal Roman dominating functions: Extensions and enumeration. Technical Report 2204.04765, Cornell University, ArXiv/CoRR, 2022. doi:10.48550/arXiv.2204.04765.

3 F. N. Abu-Khzam, H. Fernau, and K. Mann. Minimal Roman dominating functions: Extensions and enumeration. In M. A. Bekos and M. Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG*, volume 13453 of *LNCS*, pages 1–15. Springer, 2022. doi:10.1007/978-3-031-15914-5_1.

4 F. N. Abu-Khzam, H. Fernau, and K. Mann. Roman census: Enumerating and counting Roman dominating functions on graph classes. Technical Report 2208.05261, Cornell University, ArXiv/CoRR, 2022. arXiv:2208.05261.

5 F. N. Abu-Khzam and P. Heggernes. Enumerating minimal dominating sets in chordal graphs. *Information Processing Letters*, 116(12):739–743, 2016.

6 C. Bazgan, L. Brankovic, K. Casel, H. Fernau, K. Jansen, K.-M. Klein, M. Lampis, M. Liedloff, J. Monnot, and V. Paschos. The many facets of upper domination. *Theoretical Computer Science*, 717:2–25, 2018.

7 S. Benecke. Higher order domination of graphs. Master's thesis, Department of Applied Mathematics of the University of Stellebosch, South Africa, http://dip.sun.ac.za/~vuuren/Theses/Benecke.pdf, 2004.

8 S. Bermudo and H. Fernau. Computing the differential of a graph: hardness, approximability and exact algorithms. *Discrete Applied Mathematics*, 165:69–82, 2014.

9 S. Bermudo and H. Fernau. Combinatorics for smaller kernels: The differential of a graph. *Theoretical Computer Science*, 562:330–345, 2015.

10 S. Bermudo, H. Fernau, and J. M. Sigarreta. The differential and the Roman domination number of a graph. *Applicable Analysis and Discrete Mathematics*, 8:155–171, 2014.

11 T. Bläsius, T. Friedrich, J. Lischeid, K. Meeks, and M. Schirneck. Efficiently enumerating hitting sets of hypergraphs arising in data profiling. *Journal of Computer and System Sciences*, 124:192–213, 2022.

12 M. Bonamy, O. Defrain, M. Heinrich, and J.-F. Raymond. Enumerating minimal dominating sets in triangle-free graphs. In R. Niedermeier and C. Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *LIPIcs*, pages 16:1–16:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

13 D. Bród. On the number of minimal dominating sets in some classes of trees. *Int. J. Contemp. Math. Sciences*, 6:503–506, 2011.

14 K. Casel, H. Fernau, M. Khosravian Ghadikolaei, J. Monnot, and F. Sikora. Extension of some edge graph problems: Standard and parameterized complexity. In L. A. Gasieniec, J. Jansson, and C. Levcopoulos, editors, *Fundamentals of Computation Theory - 22nd International Symposium, FCT*, volume 11651 of *LNCS*, pages 185–200. Springer, 2019.

15 K. Casel, H. Fernau, M. Khosravian Ghadikolaei, J. Monnot, and F. Sikora. Abundant extensions. In T. Calamoneri and F. Corò, editors, *Algorithms and Complexity - 12th International Conference, CIAC*, volume 12701 of *LNCS*, pages 3–17. Springer, 2021. doi:10.1007/978-3-030-75242-2_1.

16 E. W. Chambers, B. Kinnersley, N. Prince, and D. B. West. Extremal problems for Roman domination. *SIAM Journal of Discrete Mathematics*, 23:1575–1586, 2009.

17 M. Chapelle, M. Cochefert, J.-F. Couturier, D. Kratsch, M. Liedloff, and A. Perez. Exact algorithms for weak Roman domination. In T. Lecroq and L. Mouchard, editors, *Combinatorial Algorithms - 24th International Workshop, IWOCA*, volume 8288 of *LNCS*, pages 81–93. Springer, 2013.

18 M. Chellali, T. W. Haynes, S. M. Hedetniemi, S. T. Hedetniemi, and A. A. McRae. A Roman domination chain. *Graphs and Combinatorics*, 32(1):79–92, 2016.

19 J.-F. Couturier, P. Heggernes, P. van 't Hof, and D. Kratsch. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. In M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science*, volume 7147 of *LNCS*, pages 202–213. Springer, 2012.

20 J.-F. Couturier, P. Heggernes, P. van 't Hof, and D. Kratsch. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. *Theoretical Computer Science*, 487:82–94, 2013.

21 J.-F. Couturier, R. Letourneur, and M. Liedloff. On the number of minimal dominating sets on some graph classes. *Theoretical Computer Science*, 562:634–642, 2015.

22 N. Creignou, M. Kröll, R. Pichler, S. Skritek, and H. Vollmer. A complexity theory for hard enumeration problems. *Discrete Applied Mathematics*, 268:191–209, 2019.

23 P. A. Dreyer. *Applications and Variations of Domination in Graphs*. PhD thesis, Rutgers University, New Jersey, USA, 2000.

24 T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

25 O. Favaron, H. Karami, R. Khoeilar, and S. M. Sheikholeslami. On the Roman domination number of a graph. *Discrete Mathematics*, 309(10):3447–3451, 2009.

26 H. Fernau. Roman Domination: a parameterized perspective. *International Journal of Computer Mathematics*, 85:25–38, 2008.

27 F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.

28 F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer, 2010.

29 A. Gainer-Dewar and P. Vera-Licona. The minimal hitting set generation problem: Algorithms and computation. *SIAM Journal of Discrete Mathematics*, 31(1):63–100, 2017.

30 P. A. Golovach, P. Heggernes, M. M. Kanté, D. Kratsch, and Y. Villanger. Enumerating minimal dominating sets in chordal bipartite graphs. *Discrete Applied Mathematics*, 199:30–36, 2016.

31 P. A. Golovach, P. Heggernes, M. Moustapha Kanté, D. Kratsch, and Y. Villanger. Minimal dominating sets in interval graphs and trees. *Discrete Applied Mathematics*, 216:162–170, 2017.

32 P. A. Golovach, P. Heggernes, and D. Kratsch. Enumerating minimal connected dominating sets in graphs of bounded chordality. *Theoretical Computer Science*, 630:63–75, 2016.

33 P. A. Golovach, D. Kratsch, M. Liedloff, and M. Y. Sayadi. Enumeration and maximum number of minimal dominating sets for chordal graphs. *Theoretical Computer Science*, 783:41–52, 2019.

34 T. W. Haynes, S.T. Hedetniemi, and M. A. Henning, editors. *Topics in Domination in Graphs*, volume 64 of *Developments in Mathematics*. Springer, 2020.

35 S. T. Hedetniemi, R. R. Rubalcaba, P. J. Slater, and M. Walsh. Few compare to the great Roman empire. *Congressus Numerantium*, 217:129–136, 2013.

36 M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal of Discrete Mathematics*, 28(4):1916–1929, 2014.

37 M. M. Kanté, V. Limouzy, A. Mary, L. Nourine, and T. Uno. Polynomial delay algorithm for listing minimal edge dominating sets in graphs. In F. Dehne, J.-R. Sack, and U. Stege, editors, *Workshop on Algorithms and Data Structures, WADS*, volume 9214 of *LNCS*, pages 446–457. Springer, 2015.

38 M. M. Kanté, V. Limouzy, A. Mary, L. Nourine, and T. Uno. A polynomial delay algorithm for enumerating minimal dominating sets in chordal graphs. In E. W. Mayr, editor, *International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2015*, volume 9224 of *LNCS*, pages 138–153. Springer, 2016.

39 T. Kraner Šumenjak, P. Pavlić, and A. Tepeh. On the Roman domination in the lexicographic product of graphs. *Discrete Applied Mathematics*, 160(13-14):2030–2036, 2012.

40 M. Liedloff. *Algorithmes exacts et exponentiels pour les problèmes NP-difficiles: domination, variantes et généralisations*. PhD thesis, Université Paul Verlaine - Metz, France, 2007.

41 M. Liedloff, T. Kloks, J. Liu, and S.-L. Peng. Efficient algorithms for Roman domination on some classes of graphs. *Discrete Applied Mathematics*, 156(18):3400–3415, 2008.

42 C.-H. Liu and G. J. Chang. Roman domination on 2-connected graphs. *SIAM Journal of Discrete Mathematics*, 26(1):193–205, 2012.

43 C.-H. Liu and G. J. Chang. Upper bounds on Roman domination numbers of graphs. *Discrete Mathematics*, 312(7):1386–1391, 2012.

44 C.-H. Liu and G. J. Chang. Roman domination on strongly chordal graphs. *Journal of Combinatorial Optimization*, 26(3):608–619, 2013.

45 A. Mary. *Énumération des dominants minimaux d'un graphe*. PhD thesis, LIMOS, Université Blaise Pascal, Clermont-Ferrand, France, November 2013.

46 J. L. Mashburn, T. W. Haynes, S. M. Hedetniemi, S. T. Hedetniemi, and P. J. Slater. Differentials in graphs. *Utilitas Mathematica*, 69:43–54, 2006.

47 B. P. Mobaraky and S. M. Sheikholeslami. Bounds on Roman domination numbers of graphs. *Matematitchki Vesnik*, 60:247–253, 2008.

48 A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D. S. Taylor, and P. Widmayer. Server placements, Roman domination and other dominating set variants. In R. A. Baeza-Yates, U. Montanari, and N. Santoro, editors, *Foundations of Information Technology in the Era of Networking and Mobile Computing, IFIP 17th World Computer Congress — TC1 Stream / 2nd IFIP International Conference on Theoretical Computer Science IFIP TCS*, pages 280–291. Kluwer, 2002. Also available as Technical Report 365, ETH Zürich, Institute of Theoretical Computer Science, 10/2001.

49 S.-L. Peng and Y.-H. Tsai. Roman domination on graphs of bounded treewidth. In *The 24th Workshop on Combinatorial Mathematics and Computation Theory*, pages 128–131, 2007.

50 G. Rote. The maximum number of minimal dominating sets in a tree. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1201–1214. SIAM, 2019.

51 W. Shang, X. Wang, and X. Hu. Roman domination and its variants in unit disk graphs. *Discrete Mathematics, Algorithms and Applications*, 2(1):99–106, 2010.

52 Z. Shi and K. M. Koh. Counting the number of minimum Roman dominating functions of a graph. Technical report, ArXiv / CoRR, abs/1403.1019, 2014. `arXiv:1403.1019`.

53 I. B. Skjørten. Faster enumeration of minimal connected dominating sets in split graphs. Master's thesis, Department of Informatics, University of Bergen, Norway, June 2017.

54 J. M. M. van Rooij. *Exact Exponential-Time Algorithms for Domination Problems in Graphs*. PhD thesis, Universiteit Utrecht, The Netherlands, 2011.

55 H.-M. Xing, X. Chen, and X.-G. Chen. A note on Roman domination in graphs. *Discrete Mathematics*, 306(24):3338–3340, 2006.

56 F. Xueliang, Y. Yuansheng, and J. Baoqi. Roman domination in regular graphs. *Discrete Mathematics*, 309(6):1528–1537, 2009.

57 I. G. Yero and J. A. Rodríguez-Velázquez. Roman domination in Cartesian product graphs and strong product graphs. *Applicable Analysis and Discrete Mathematics*, 7:262–274, 2013.