

ROMAN DOMINATION: a parameterized perspective

Henning Fernau, Theoretische Informatik, Universität Trier, Germany,
`fernau@informatik.uni-trier.de`

(March 2006)

We analyze the graph-theoretic formalization of ROMAN DOMINATION, dating back to the military strategy of Emperor Constantine, from a parameterized perspective. More specifically, we prove that this problem is $W[2]$ -complete for general graphs. However, parameterized algorithms are presented for graphs of bounded treewidth and for planar graphs. Moreover, it is shown that a parametric dual of ROMAN DOMINATION is in \mathcal{FPT} .

1 Introduction

ROMAN DOMINATION is one of the many variants of dominating set problems [7, 12, 17]. It comes with a nice (hi)story: namely, it should reflect the idea of how to secure the Roman Empire by positioning the armies (legions) on the various parts of the Empire in a way that either (1) a specific region r is also the location of at least one army or (2) one region r' neighboring r has two armies, so that r' can afford sending off one army to the region r (in case of an attack) without loosing self-defense capabilities.

More specifically, Emperor Constantine had a look at the map of Fig. 1 or a variant thereof (as discussed in [23]). The historical background is also nicely described in the online John Hopkins Magazine, more specifically, visit <http://www.jhu.edu/~jhumag/0497web/locate3.html>. This problem is similar to the island hopping strategy pursued by General MacArthur in World War II in the Pacific theater to gradually increase the US-secured areas.

A good overview on problems related to Roman domination can be found in [2]. We assume that solving algorithms similar to the ones presented in this paper can be also found for most of these variants, in particular regarding multi-attack variants [8, 16, 18, 19]. Efficient algorithms for various graph

A preliminary version of this paper appeared in the Proc. of SOFSEM 2006.

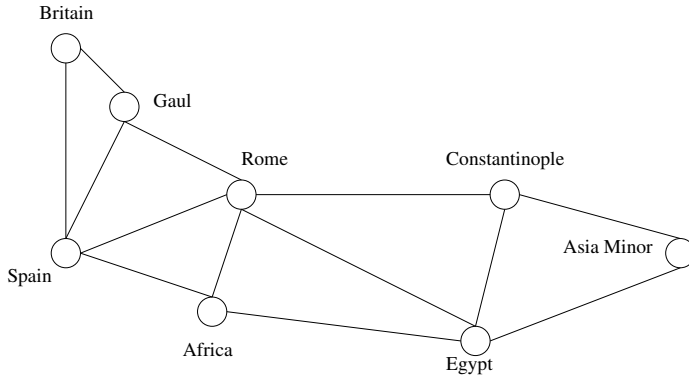


Figure 1. The Roman Empire in the times of Constantine

classes have been presented in [12,21]. Relations with the concrete problem under consideration and (more practical) network problems have been exhibited in [22].

The paper is organized as follows: In the following section, the necessary definitions are provided. Subsequently, we prove that ROMAN DOMINATION (on general graphs) is parameterized intractable. Then, we show how some structural results derived for another problem on planar graphs can be used to efficiently solve PLANAR ROMAN DOMINATION, i.e., to solve the problem in fixed-parameter time by search tree techniques. Finally, we explain how to solve ROMAN DOMINATION on graphs of bounded treewidth; this also allows for a different approach to solve PLANAR ROMAN DOMINATION, as we discuss.

2 Definitions

Let us first formally describe the problem. To this end, notice that we will use standard notions from graph theory. Throughout the paper, we deal with simple undirected graphs. $N(v)$ is the open neighborhood of vertex v , and $N[v] = N(v) \uplus \{v\}$ is the closed neighborhood, where \uplus denotes disjoint set union. An instance of ROMAN DOMINATION (ROMAN) is given by a graph $G = (V, E)$, and the parameter, a positive integer k . The question is: Is there a *Roman domination* function R such that $R(V) := \sum_{x \in V} R(x) \leq k$?

Here, a *Roman domination* function of a graph $G = (V, E)$ is a function $R : V \rightarrow \{0, 1, 2\}$ with

$$\forall v \in V : R(v) = 0 \Rightarrow \exists x \in N(v) : R(x) = 2.$$

$D_R = R^{-1}(\{1, 2\})$ is then the *Roman domination set*. The minimum of $R(V)$

over all valid Roman domination functions R is also called the *Roman domination number* of a given graph.

In the following, we give the necessary background on parameterized complexity: A *parameterized problem* P is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a fixed alphabet and \mathbb{N} is the set of all non-negative integers. Therefore, each instance of the parameterized problem P is a pair (I, k) , where the second component k is called the *parameter*. The language $L(P)$ is the set of all YES-instances of P . We say that the parameterized problem P is *fixed-parameter tractable* [11] if there is an algorithm (realizable by a deterministic Turing machine) that decides whether an input (I, k) is a member of $L(P)$ in time $f(k)|I|^c$, where c is a fixed constant and f is a function whose argument k is independent of the overall input length $|I|$. The class of all fixed-parameter tractable problems is denoted by \mathcal{FPT} .

The $\mathcal{O}^*(\cdot)$ notation has by now become standard in exact algorithms. It is meant to not only suppress constants (as the more familiar $\mathcal{O}(\cdot)$ -notation does) but also polynomial parts in the function estimates. Hence, a problem is in \mathcal{FPT} iff an instance (with parameter k) can be solved in time $\mathcal{O}^*(f(k))$ for some function f .

There is also a hardness theory, most notably, the $W[t]$ hierarchy, that complements fixed-parameter tractability:

$$\mathcal{FPT} = W[0] \subseteq W[1] \subseteq W[2] \subseteq \dots$$

It is commonly believed that this hierarchy is strict. Since only the second level $W[2]$ will be of interest to us in this paper, we will only define that class below. We do this in the ‘‘Turing way’’ as (partially) followed in [4–6, 14].

A *parameterized reduction* is a function r that, for some polynomial p and some function g , is computable in time $\mathcal{O}(g(k)p(|I|))$ and maps an instance (I, k) of \mathcal{P} onto an instance $r(I, k) = (I', k')$ of \mathcal{P}' such that (I, k) is a YES-instance of \mathcal{P} if and only if (I', k') is a YES-instance of \mathcal{P}' and $k' \leq g(k)$. We also say that \mathcal{P} *reduces to* \mathcal{P}' .

$W[2]$ can be characterized by the following problem on Turing machines:

An instance of SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION (SMNTMC) is given by a multi-tape nondeterministic Turing machine M (with two-way infinite tapes), an input string x , and the parameter, a positive integer k . The question is: Is there an accepting computation of M on input x that reaches a final accepting state in at most k steps?

More specifically, a parameterized problem is in $W[2]$ iff it can be reduced with a parameterized reduction to SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION, see [4].

3 ROMAN DOMINATION on general graphs is hard

LEMMA 3.1 ROMAN DOMINATION *is in* $W[2]$.

Proof Let $G = (V, E)$ be an instance of ROMAN DOMINATION. We have to transform it into an instance of SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION. We also assume that $k > 0$ ($k = 0$ is a trivial instance).

The corresponding Turing machine T has $|V| + 1$ tapes; let them be indexed by $\{0\} \cup V$. As tape symbols, we will use $(V \times \{1, 2\})$ on tape 0 and $\#$ on the other tapes (besides the blank symbol). The edge relation of G is “hard-wired” into the transition function of T as described below. The input string is empty.

In a first phase, T nondeterministically guesses the Roman domination function R and writes it on tape 0 using the letters from $V \times \{1, 2\}$ as follows: T moves the head on tape 0 one step to the right, and writes there a guess $(v, i) \in (V \times \{1, 2\})$. Upon writing (v, i) , T also increments an internal-memory counter c by i . As long as $c \leq k$, T can nondeterministically continue in phase one or transition into phase two; if $c > k$, T hangs up.

In a second phase, T has to verify that the previous guesses are correct. To this end, upon reading symbol $(v, 1)$ on tape 0, T writes $\#$ on the tape addressed by v and moves that head one step to the right. Upon reading $(v, 2)$ on tape 0, T writes $\#$ on all tapes addressed by vertices from $N[v]$ and moves the corresponding heads one step to the right. Moreover, after reading symbol (v, i) on tape 0, T moves the head on tape 0 one step to the left. Upon reading the blank symbol on tape 0, T moves all other heads one step to the left; only if then all V -addressed tapes show $\#$ under their respective heads, T accepts. The second phase will take another $k + 1$ steps.

It is now easy to see that (G, k) is a YES-instance to ROMAN DOMINATION iff T has an accepting computation within $2k + 1$ steps, so that we actually described a parameterized reduction. \square

We will show $W[2]$ -hardness with the help of the following problem: An instance of RED-BLUE DOMINATING SET (RBDS) is given by a graph $G = (V, E)$ with V partitioned as $V_{\text{red}} \uplus V_{\text{blue}}$, and the parameter, a positive integer k . The question is: Is there a *red-blue dominating set* $D \subseteq V_{\text{red}}$ with $|D| \leq k$, i.e., $V_{\text{blue}} \subseteq N(D)$?

We need the following result, that can be easily distilled from [11]:

LEMMA 3.2 RED-BLUE DOMINATING SET, RESTRICTED TO BIPARTITE GRAPHS *is* $W[2]$ -hard.

To prove the hardness result, we need one fact about the Roman domination of complete graphs that follows from [7, Prop. 9].

LEMMA 3.3 *For the complete graph K_n on n vertices, the Roman domination*

number is two iff $n \geq 2$.

THEOREM 3.4 ROMAN DOMINATION is $W[2]$ -complete.

Proof By Lemma 3.1, we already know that ROMAN DOMINATION lies in $W[2]$.

Assume that $G = (V, E)$ is an instance of RED-BLUE DOMINATING SET, RESTRICTED TO BIPARTITE GRAPHS (see Lemma 3.2), i.e., $V = V_{\text{red}} \uplus V_{\text{blue}}$. W.l.o.g., we can assume that $|V_{\text{red}}| > 1$. In the simulating ROMAN DOMINATION instance, we construct a graph $G' = (V', E')$, where

$$V' = (V_{\text{red}} \cup \{1, \dots, 2k + 1\}) \times \{1, \dots, k\} \cup V_{\text{blue}},$$

and E' contains the following edges (and no others):

- (i) $G'[V_{\text{red}} \times \{i\}]$ is a complete graph for each $i \in \{1, \dots, k\}$.
- (ii) For all $i \in \{1, \dots, k\}$ and $x \in V_{\text{red}}, y \in V_{\text{blue}}, \{x, y\} \in E$ iff $\{[x, i], y\} \in E'$.
- (iii) For all $i \in \{1, \dots, k\}, j \in \{1, \dots, 2k + 1\}$ and $x \in V_{\text{red}}: \{[x, i], [j, i]\} \in E'$.

We are going to show the following claim: G has a red-blue dominating set D of size k iff G' has a Roman domination function R with $\sum_{x \in D_R} R(x) = 2k$.

If G has a red-blue dominating set $D = \{d_1, \dots, d_k\}$ of size k , then consider the following function $R : V' \rightarrow \{0, 1, 2\}$: R assigns zero to all vertices but to $d'_i = [d_i, i]$, to which R assigns two. Since d'_i is connected to all vertices in $(V_{\text{red}} \cup \{1, \dots, 2k + 1\}) \times \{i\}$, the vertices in $V' \setminus V$ are all dominated by this assignment. Moreover, since D is a red-blue dominating set of G , all vertices in V_{blue} are dominated in G' , as well.

Now consider a Roman domination function R for G' with $\sum_{x \in D_R} R(x) = 2k$. Due to Lemma 3.3 and according to the first condition on edges, the Roman domination number of each induced graph $G'[V_{\text{red}} \times \{i\}]$ is two, assuming $|V_{\text{red}}| > 1$. Since $G'[V_{\text{red}} \times \{1, \dots, k\}]$ can be decomposed into k components, the Roman domination number of $G'[V_{\text{red}} \times \{1, \dots, k\}]$ is $2k$. More specifically, to achieve that bound, the domination function would have to assign two to one vertex from $V_{\text{red}} \times \{i\}$ for each i and zero to all other vertices. Observe that such an assignment would be also a valid Roman domination function R' for $G'[(V_{\text{red}} \cup \{1, \dots, 2k + 1\}) \times \{1, \dots, k\}]$ if we assign zero to all vertices from $\{1, \dots, 2k + 1\} \times \{1, \dots, k\}$.

Since there are “too many” vertices in $\{1, \dots, 2k + 1\} \times \{1, \dots, k\}$, we cannot simply replace one or more vertices to which R' assigns two by vertices from $\{1, \dots, 2k + 1\} \times \{1, \dots, k\}$ to which R' (as constructed) had assigned zero.

Observe that we have left over yet some degrees of freedom for finally constructing a valid Roman domination function R from R' ; namely, we have not been specific about how to choose a vertex from $V_{\text{red}} \times \{i\}$ (for each i) to which we assign two. However, if we find k assignments of two to vertices from $V_{\text{red}} \times \{1, \dots, k\}$ such that also all vertices from V_{blue} are dominated,

i.e., $D_R = \{[d_1, 1], \dots, [d_k, k]\} = R^{-1}(\{2\})$, then $D = \{d_1, \dots, d_k\}$ is a valid dominating set of G .

Since there are no edges between vertices from $\{1, \dots, 2k+1\} \times \{1, \dots, k\}$ and V_{blue} , there is no way of replacing some of the vertices selected from $(V_{\text{red}} \cup \{1, \dots, 2k+1\}) \times \{1, \dots, k\}$ (by assigning two to them) by vertices from V_{blue} , so that there cannot be a Roman domination function R that assigns one or two to any of the vertices from V_{blue} without violating the condition $\sum_{x \in D_R} R(x) = 2k$. So, the Roman domination function as constructed above is the only possibility; that construction works if and only if G has a dominating set of size k . \square

The previous Theorem also sharpens [12, Theorem 2.42].

We conclude this section by briefly discussing one further problem, also taken from [22]; in fact, some more (and similar) problems can be found there and treated alike. An instance of DOMINATING REARRANGEMENT (DR) is given by a graph $G = (V, E)$, a subset $S \subseteq V$, and the parameter, a positive integer $k = |S|$. The question is: Is there a *dominating rearrangement* $r : S \rightarrow N[S], s \mapsto r(s) \in N[s]$ such that $r(S) \subseteq V$ is a dominating set?

Again, this problem can be viewed from a military perspective: S is the set of locations where currently armies are placed on, and the question is if by a one-step rearrangement of each army (if necessary) a situation can be created in which each region (modeled by graph vertices) is sheltered by either a defending army in the region itself or in a neighboring region.

This problem is interesting for at least two reasons from the perspective of parameterized complexity:

- The parameterization is not arising from an optimization problem.
- The problem can be viewed as a *local search problem*, parameterized by a given “temporary” solution. Such type of problems can show up in many disguises in practice.

THEOREM 3.5 DOMINATING REARRANGEMENT is $W[2]$ -complete.

Proof Membership in $W[2]$ can be seen by a guess-and-verify strategy as seen in the proof of Lemma 3.1. For the hardness, take again an instance $(G = (V = V_{\text{red}} \uplus V_{\text{blue}}, E), k)$ of RED-BLUE DOMINATING SET. Let $S = \{1, \dots, k\}$ be disjoint from V , and consider the graph $G' = (V', E')$ with $V' = V \cup S$ and $E' = E \cup (S \times V_{\text{red}})$. Hence, $G'[S \cup V_{\text{red}}]$ forms a complete bipartite graph. This gives the instance (G', S) of DOMINATING REARRANGEMENT. Obviously, $D \subseteq V_{\text{red}}$ is a dominating set of size (at most) k iff (G', S) can be solved by moving $|D|$ of the armies in S onto the vertices from D . \square

We will not discuss DOMINATING REARRANGEMENT in the following sections but only mention here that the problem can be put in \mathcal{FPT} when restricted

to planar graph instances by the bounded treewidth techniques, although we do not know of a direct search tree algorithm as presented for ROMAN DOMINATION in the next section.

4 ROMAN DOMINATION on planar graphs

From a historical perspective, this is somehow the “original” problem, indeed: taking a map of the Roman Empire and assuming firstly that different regions are interpreted as vertices of a graph and finally that regions are neighbored if they share a common borderline (as opposed to having boundaries meeting in a single point), then this neighborhood (multi-)graph is (as the geometric dual of the map) planar.

We will first sketch a search tree algorithm that puts PLANAR ROMAN DOMINATION into \mathcal{FPT} . From the standpoint of parameterized algorithmics, this is an interesting algorithm, since it “recycles” most of the rules and terminology that was earlier developed for PLANAR DOMINATING SET in [1, 14].

There, we introduced the notion of a *black and white graph*. The vertex set V of G is partitioned into two disjoint sets B and W of *black* and *white* vertices, respectively, i.e., $V = B \uplus W$. Black vertices are those vertices which still need to be dominated, while white vertices are already dominated, but it is still possible to place two armies on such a vertex in order to protect the neighboring vertices. In each step of the search tree, we would like to branch according to a low degree black vertex.

Formally, this means that we solve an annotated version of ROMAN DOMINATION, namely on black and white graphs. We propose to use the following reduction rules:

- (R1) Delete an edge between white vertices.
- (R2) Delete a pendant white vertex, i.e., a vertex of degree one.
- (R4) If there is a white vertex u of degree 2, with two black neighbors u_1 and u_2 connected by an edge $\{u_1, u_2\}$, then delete u .
- (R5) If there is a white vertex u of degree 2, with black neighbors u_1, u_3 , and there is a black vertex u_2 and edges $\{u_1, u_2\}$ and $\{u_2, u_3\}$ in G , then delete u .
- (R6) If there is a white vertex u of degree 2, with black neighbors u_1, u_3 , and there is a white vertex u_2 and edges $\{u_1, u_2\}$ and $\{u_2, u_3\}$ in G , then delete u .
- (R7) If there is a white vertex u of degree 3, with black neighbors u_1, u_2, u_3 for which the edges $\{u_1, u_2\}$ and $\{u_2, u_3\}$ are present in G (and possibly also $\{u_1, u_3\}$), then delete u .

The peculiar numbering is in accordance with our rule numbering scheme for PLANAR DOMINATING SET in [14] and should make clear that we actually must only replace one of the rules with some additional branching in our algorithm,

in order to get rid of pendant black vertices.

LEMMA 4.1 *The reduction rules are sound.*

Proof (R1), (R2) are immediate.

(R4): Let $G = (V, E)$ be a black and white graph and $G' = (V', E')$ be obtained from G by applying (R4) once. Hence, there are vertices u, u_1, u_2 in V as described in (R4). If R' is a valid Roman domination function of G' , then R' can be extended to a valid Roman domination function R' on V by setting $R'(u) = 0$. Obviously, $R'(V') = R'(V)$. If R is a valid Roman domination function of G , then R restricted to V' will be valid if $R(u) = 0$. Then, $R(V') = R(V)$. The case $R(u) = 1$ need not be considered, since u is white. If $R(u) = 2$, then $R(u_1) + R(u_2) \leq 1$, since otherwise by redefining $R(u) := 0$ a smaller valid Roman domination function can be obtained. However, if $R(u) \leq 1$, then $R(u_1) = 0$ or $R(u_2)$. Assuming $R(u_1) = 0$, we can obtain a valid Roman domination function by setting $R(u) := 0$ and $R(u_1) := 2$ without changing the overall value. Hence, after the indicated modifications, R restricted to V' will be valid, and $R(V') = R(V)$.

(R5), (R6), (R7) can be argued in a similar fashion. \square

Remark 1 As can be seen by the proof for rule (R4) (and similar for rules (R5), (R6), (R7)), the argument for its validity can be paraphrased as: an optimum dominating set that contains the white degree-two vertex u can be turned into another optimum dominating set by replacing u by one of its neighbors. This rule is not valid for DOMINATING REARRANGEMENT, since u might have had higher degree in the original graph (for example, rule (R1) decreases the degree of white vertices), so that the only possible rearrangement would be to put u into the dominating set.

In other words, it is unknown if all minimal dominating sets of size at most k can be enumerated for planar graphs in \mathcal{FPT} -time (see [9, 13]); otherwise, we could use this methodology to solve DOMINATING REARRANGEMENT on planar graphs.

A careful check of the reduction rules as developed for PLANAR DOMINATING SET show that all are valid but one, namely rule (R3) in [14], which is dealing with a black vertex x of degree one (it is not clear if one army should be put on x or two armies on the neighbor of x). That particular rule is not used in the (non-trivial) proof of the following theorem from [1, 14], where “reduced” refers to all reduction rules from [14] but (R3).

THEOREM 4.2 *If $G = (B \uplus W, E)$ is a planar black and white graph that is reduced, then there exists a black vertex $u \in B$ with $\deg_G(u) \leq 7$.*

A simple search tree algorithm would now pick a black vertex v of smallest degree and branch according to if $R(v) = 1$ or if $R(u) = 2$ for some $u \in$

$N[v]$; this branching reduces the parameter by two for each u ; according to Thm. 4.2, $N[v]$ contains at most eight vertices. As to the black and white coloring, initially all vertices would be black. When branching at vertex v , putting $R(v) = 1$ would mean to simply delete v from the instance. If $R(u) = 2$ for some $u \in N[v]$, then u would be deleted from the instance and all vertices in $N(u)$ would be colored white. Solving the corresponding recurrence $T(k) \leq T(k-1) + 8T(k-2)$ for the size of the search tree shows the following assertion:

THEOREM 4.3 PLANAR ROMAN DOMINATION *can be solved in $\mathcal{O}^*(3.3723^k)$ time.* \square

5 ROMAN DOMINATION on graphs of bounded treewidth

In this section, we reconsider the problem of determining the minimum Roman domination set on graphs of bounded treewidth. This problem has been previously attacked in [22], but their algorithm is not quite correct, as we will explain. Then, we apply this treewidth-based algorithm to obtain $\mathcal{O}(c^{\sqrt{k}})$ algorithms for PLANAR ROMAN DOMINATION. Details on tree decompositions can be found in [20]. On graphs of bounded treewidth, many otherwise combinatorially hard problems can be efficiently solved by dynamic programming.

Let us briefly recall some details on tree decompositions:

Definition 5.1 Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\langle \{X_i \mid i \in I\}, T \rangle$, where each X_i is a subset of V , called a *bag*, and T is a tree with the elements of I as nodes. The following three properties must hold:

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

The *width* of the tree decomposition $\langle \{X_i \mid i \in I\}, T \rangle$ equals

$$\max\{|X_i| \mid i \in I\} - 1.$$

The *treewidth* of G is the minimum k such that G has a tree decomposition of width k , also written $\text{tw}(G)$ for short.

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

Definition 5.2 A tree decomposition $\langle \{X_i \mid i \in I\}, T \rangle$ with a distinguished

root node r is called a *nice tree decomposition* if the following conditions are satisfied:

- (i) Every node of the tree T has at most 2 children.
- (ii) If a node N has two children N' and N'' , then $X_N = X_{N'} = X_{N''}$ (in this case N is called a *join node*).
- (iii) If a node N has one child N' , then either
 - a) $|X_N| = |X_{N'}| + 1$ and $X_{N'} \subset X_N$ (in this case N is called an *insert node* or an *introduce node*), or
 - b) $|X_N| = |X_{N'}| - 1$ and $X_N \subset X_{N'}$ (in this case N is called a *forget node*).

Observe that each node in a nice tree decomposition is either a join node, an insert node, a forget node, or a leaf node.

If the tree in a tree decomposition happens to be a path, we also speak of a *path decomposition*, thus defining the notion of *pathwidth*.

Given a nice tree decomposition of a graph, we have hence to specify the operations in four different types of nodes, see [3, 20, 24]. In the dynamic programming, to each node a table is associated that stores all possible combinations of “vertex states” together with their optimal value. With ROMAN DOMINATION, we need to store four states per vertex (only three are used in [22]):

- 0,1,2 are the values that the Roman domination function is assumed to assign to a particular vertex.
- $\hat{0}$ also tells us that the Roman domination function assigns 0 to that vertex.

The difference in the semantics of $0, \hat{0}$ is the following: the assignment of 0 means that the vertex is already dominated at the current stage of the algorithm, and $\hat{0}$ means that, at the current stage of the algorithm, we still ask for a domination of this vertex.

To fix notation, the optimal value of a table in node N (with bag X_N) at a concrete assignment $\vec{x} \in \{0, \hat{0}, 1, 2\}^{|X_N|}$ is denoted by $N(\vec{x})$. We will view \vec{x} also as a mapping $X_N \rightarrow \{0, \hat{0}, 1, 2\}$ whenever convenient. We associate to \vec{x} the mapping R_N that identifies 0 and $\hat{0}$, i.e., $R_N(\vec{x})(v) = \vec{x}(v)$ if $\vec{x}(v) \neq \hat{0}$ and $R_N(\vec{x})(v) = 0$ if $\vec{x}(v) = \hat{0}$. Define $|\vec{x}|_N = \sum_{v \in X_N} R_N(\vec{x})(v)$ to be the assumed value of the Roman domination function within X_N .

For example, the graph from Fig. 1 has pathwidth two, as can be seen in Fig. 2. To actually get a nice pathdecomposition, one has to intercalate intermediate bags with two vertices between two explicitly given bags. For example, between the blue bag containing Britain, Spain and Gaul, and the green bag with Gaul, Spain and Rome, an intermediate bag with Gaul and Spain would come. Let us explain how the initialization and the processing of the forget and join nodes is done with this example.

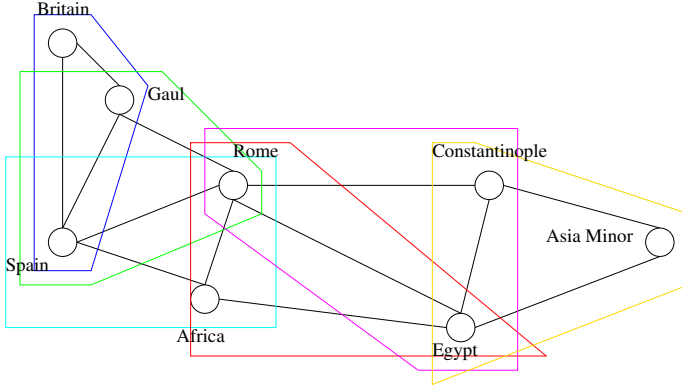


Figure 2. A graph of pathwidth two.

Initialization. Let N be a leaf node of the given tree decomposition. Then, for each $\vec{x} \in \{0, \hat{0}, 1, 2\}^{X_N}$, let $N(\vec{x}) = |\vec{x}|_N$ whenever \vec{x} is a feasible initial assignment, i.e.,

- whenever $\vec{x}(v) = \hat{0}$ for some $v \in X_N$, then there is no $v' \in X_N$ that is neighbor to v such that $\vec{x}(v') = 2$, and
- whenever $\vec{x}(v) = 0$ for some $v \in X_N$, then there is some $v' \in X_N$ that is neighbor to v such that $\vec{x}(v') = 2$.

If the assignment \vec{x} is not feasible, then we set $N(\vec{x}) = \infty$, this way indicating an error case. The initial table for the bag containing Britain (B), Gaul (G), and Spain (S) is displayed in Table 1.

Forget node processing. Let N be a forget node, i.e., N has a child N' with $X_{N'} = X_N \cup \{v_{old}\}$. Then,

$$N(\vec{x}) = \min\{N'(\vec{x}') \mid \forall v \in N : \vec{x}'(v) = \vec{x}(v) \wedge \vec{x}'(v_{old}) \in \{0, 1, 2\}\}.$$

Namely, it is not feasible to “forget about” the value $\hat{0}$ of v_{old} since by the conditions posed on a tree decomposition, v_{old} cannot be dominated by a neighbor with two armies assigned later on. If we forget Britain in our example, this leads to Table 2:

Insert node processing. Let N be an insert node, i.e., N has a child N' with $X_N = X_{N'} \cup \{v_{new}\}$. Let \vec{x}' denote the projection of \vec{x} onto $X_{N'}$. Then:

- If $\vec{x}(v_{new}) = 0$, then $N(\vec{x}) = N'(\vec{x}')$ if there is a neighbor $v' \in X_{N'}$ of v_{new} such that $\vec{x}'(v') = 2$; otherwise, $N(\vec{x}) = \infty$.
- If $\vec{x}(v_{new}) = \hat{0}$, then $N(\vec{x}) = N'(\vec{x}')$ if there is no neighbor $v' \in X_{N'}$ of v_{new} such that $\vec{x}'(v') = 2$; otherwise, $N(\vec{x}) = \infty$.
- If $\vec{x}(v_{new}) = 1$, then $N(\vec{x}) = N'(\vec{x}') + 1$.

B	G	S	min	B	G	S	min	B	G	S	min	B	G	S	min
0	0	0	∞	0	0	$\hat{0}$	∞	0	0	1	∞	0	0	2	2
$\hat{0}$	0	0	∞	$\hat{0}$	0	$\hat{0}$	∞	$\hat{0}$	0	1	∞	$\hat{0}$	0	2	∞
1	0	0	∞	1	0	$\hat{0}$	∞	1	0	1	∞	1	0	2	3
2	0	0	2	2	0	$\hat{0}$	∞	2	0	1	3	2	0	2	4
0	$\hat{0}$	0	∞	0	$\hat{0}$	$\hat{0}$	∞	0	$\hat{0}$	1	∞	0	$\hat{0}$	2	∞
$\hat{0}$	$\hat{0}$	0	∞	$\hat{0}$	$\hat{0}$	$\hat{0}$	0	$\hat{0}$	$\hat{0}$	1	1	$\hat{0}$	$\hat{0}$	2	∞
1	$\hat{0}$	0	∞	1	$\hat{0}$	$\hat{0}$	1	1	$\hat{0}$	1	2	1	$\hat{0}$	2	∞
2	$\hat{0}$	0	∞	2	$\hat{0}$	$\hat{0}$	∞	2	$\hat{0}$	1	∞	2	$\hat{0}$	2	∞
0	1	0	∞	0	1	$\hat{0}$	∞	0	1	1	∞	0	1	2	3
$\hat{0}$	1	0	∞	$\hat{0}$	1	$\hat{0}$	1	$\hat{0}$	1	1	2	$\hat{0}$	1	2	∞
1	1	0	∞	1	1	$\hat{0}$	2	1	1	1	3	1	1	2	4
2	1	0	3	2	1	$\hat{0}$	∞	2	1	1	4	2	1	2	5
0	2	0	2	0	2	$\hat{0}$	∞	0	2	1	3	0	2	2	4
$\hat{0}$	2	0	∞	$\hat{0}$	2	$\hat{0}$	∞	$\hat{0}$	2	1	∞	$\hat{0}$	2	2	∞
1	2	0	3	1	2	$\hat{0}$	∞	1	2	1	4	1	2	2	5
2	2	0	4	2	2	$\hat{0}$	∞	2	2	1	5	2	2	2	6

Table 1. The initial assignment in the bag $\{B, G, S\}$.

G	S	min	G	S	min	G	S	min	G	S	min
0	0	2	0	$\hat{0}$	∞	0	1	3	0	2	2
$\hat{0}$	0	∞	$\hat{0}$	$\hat{0}$	1	$\hat{0}$	1	2	$\hat{0}$	2	∞
1	0	3	1	$\hat{0}$	2	1	1	3	1	2	3
2	0	2	2	$\hat{0}$	∞	2	1	3	2	2	4

Table 2. Forgetting Britain.

- If $\vec{x}(v_{new}) = 2$, then $N(\vec{x}) = \min\{N'(\vec{y} + 2) \mid R_{N'}(x') = R_{N'}(y)\}$ if there is no neighbor $v' \in X_{N'}$ of v_{new} such that $\vec{x}(v') = \hat{0}$; otherwise, $N(\vec{x}) = \infty$.

These rules yield Table 3:

THEOREM 5.3 MINIMUM ROMAN DOMINATION, *parameterized by the pathwidth $\text{pw}(G)$ of the input graph G , can be solved in time $\mathcal{O}(4^{\text{pw}(G)}|V(G)|)$.*

Let us now point to the following additional complication when dealing with join nodes: if we update an assignment that maps vertex x onto 0, it is not necessary that both children assign 0 to x ; it is sufficient that one of the two branches does, while the other assigns $\hat{0}$. So, while it is clear that in the initialization phase and when processing insert or forget nodes, $\mathcal{O}(4^{\text{tw}(G)})$ time is needed to perform the other node actions. This also implies a $\mathcal{O}^*(4^k)$ algorithm for solving ROMAN DOMINATION on graphs whose pathwidth is bounded by k .

A naive implementation of what we said in the previous sentence would

G	S	R	min	G	S	\hat{O}	min	G	S	R	min	G	S	R	min
0	0	0	∞	0	0	\hat{O}	2	0	0	1	3	0	0	2	3
\hat{O}	0	0	∞	\hat{O}	0	\hat{O}	∞	\hat{O}	0	1	∞	\hat{O}	0	2	∞
1	0	0	∞	1	0	\hat{O}	3	1	0	1	4	1	0	2	4
2	0	0	2	2	0	\hat{O}	∞	2	0	1	3	2	0	2	4
0	\hat{O}	0	∞	0	\hat{O}	\hat{O}	∞	0	\hat{O}	1	∞	0	\hat{O}	2	∞
\hat{O}	\hat{O}	0	∞	\hat{O}	\hat{O}	\hat{O}	1	\hat{O}	\hat{O}	1	2	\hat{O}	\hat{O}	2	∞
1	\hat{O}	0	∞	1	\hat{O}	\hat{O}	1	1	\hat{O}	1	2	1	\hat{O}	2	∞
2	\hat{O}	0	∞	2	\hat{O}	\hat{O}	∞	2	\hat{O}	1	∞	2	\hat{O}	2	∞
0	1	0	∞	0	1	\hat{O}	3	0	1	1	4	0	1	2	4
\hat{O}	1	0	∞	\hat{O}	1	\hat{O}	2	\hat{O}	1	1	3	\hat{O}	1	2	∞
1	1	0	∞	1	1	\hat{O}	3	1	1	1	4	1	1	2	5
2	1	0	3	2	1	\hat{O}	∞	2	1	1	4	2	1	2	5
0	2	0	2	0	2	\hat{O}	∞	0	2	1	3	0	2	2	4
\hat{O}	2	0	∞	\hat{O}	2	\hat{O}	∞	\hat{O}	2	1	∞	\hat{O}	2	2	∞
1	2	0	3	1	2	\hat{O}	∞	1	2	1	4	1	2	2	5
2	2	0	4	2	2	\hat{O}	∞	2	2	1	5	2	2	2	6

Table 3. Inserting Rome.

amount in spending $\mathcal{O}(16^{\text{tw}(G)})$ time for the join node processing. However, the “monotonicity trick” observed in [1] also works for this problem. In order to make this trick work, we have to re-interpret the assignment \hat{O} to a vertex v as follows: it should now mean that v is assigned zero by the domination function that is constructed but that it is not known if v is already dominated or not. With this slightly changed semantics, the following claim can be made: If \vec{x} and \vec{y} are two assignments for node N in a tree decomposition and if $R_N(\vec{x}) = R_N(\vec{y})$ and if

$$\forall v \in X_N : \vec{x}(v) = 0 \Rightarrow \vec{y}(v) = 0, \quad (1)$$

then $N(\vec{x}) \leq N(\vec{y})$. Namely, if $\vec{x}(v) = \hat{O}$ and $\vec{y}(v) = 0$, then the forced domination of v according to \vec{y} would possibly require resources that are not needed to satisfy the assignment \vec{x} . Due to the claim, we also write $\vec{x} \leq \vec{y}$ if $R_N(\vec{x}) = R_N(\vec{y})$ and if Property (1) is satisfied.

The rules for computing the assignments for the parent node from the children node(s) could be adopted to directly cope with this new interpretation of \hat{O} . For example, the initialization would be simplified as follows: for each $\vec{x} \in \{0, \hat{O}, 1, 2\}^{|X_N|}$, let $N(\vec{x}) = |\vec{x}|_N$ whenever \vec{x} is a feasible initial assignment, i.e., whenever $\vec{x}(v) = 0$ for some $v \in X_N$, then there is some $v' \in X_N$ that is neighbor to v such that $\vec{x}(v') = 2$.

Improved join node processing. Now, for every vertex v in the parent bag N , we consider the following cases:

- either 2, 1 or $\hat{0}$ is assigned to v ; then, the same assignment must have been made in the two children N' and N'' ;
- or 0 is assigned to v ; then, we have two possible assignments in the child nodes: 0 to v in the left child and $\hat{0}$ to v in the right child or vice versa.

Summarizing, the table entry $N(\vec{x})$ can be computed as

$$\min\{N'(\vec{y}) + N''(\vec{y}') - |\vec{x}|_N \mid \vec{y} \leq \vec{x}, \forall v \in X_N : \vec{x}(v) = 0 \implies \vec{y}(v) \neq \vec{y}'(v)\}.$$

THEOREM 5.4 MINIMUM ROMAN DOMINATION, *parameterized by the treewidth $\text{tw}(G)$ of the input graph G , can be solved in time $\mathcal{O}(5^{\text{tw}(G)}|V(G)|)$.*

Proof It only remains to argue for the claimed run time bound. As argued above, the case is even better if we consider the initialization phase or forget nodes or insert nodes. So, assume we have to compute the values in a parent join node N with k vertices. We claim that the number $T(k)$ of table entries in the child nodes one has to visit in order to compute the whole table for N satisfies $T(k) \leq 5^k$ for $k \geq 1$. If $k = 1$, the claim is trivially true. Assume $k > 1$ in what follows. For a particular assignment \vec{x} of N , consider a specific vertex $v \in X_N$. If $\vec{x}(v) \in \{0, 1, 2\}$, we have to check assignments y in the child nodes with $\vec{y}(v) = \vec{x}(v)$. The number of those assignments is the same as if v would not show up in neither N nor in the child nodes in each of the three cases. If $\vec{x}(v) = \hat{0}$, we have to check assignments y, y' in the two child nodes with $\vec{y}(v) = 0$ and $\vec{y}'(v) = \hat{0}$ and vice versa. The number of those assignments is the same as if v would not show up in neither N nor in the child nodes in each of the two cases. This reasoning shows that $T(k) \leq 5T(k-1)$, from which the claim follows by induction. \square

This also generalizes Dreyer's result on trees [12, Sec. 2.9]. Besides having a corrected version of the \mathcal{PTAS} for MINIMUM ROMAN DOMINATION explained in [22], we can also state an $\mathcal{O}^*(c^{\sqrt{k}})$ algorithm for PLANAR ROMAN DOMINATION. To get such an algorithm, we link ROMAN DOMINATION with DOMINATING SET:

LEMMA 5.5 *If $D \subseteq V$ is a Roman domination set for $G = (V, E)$ (with respect to a Roman domination function R , i.e., $D = D_R$), then D is also a dominating set. Moreover, if $\sum_{x \in D_R} R(x) \leq k$, then $|D| \leq k$.*

THEOREM 5.6 [Fomin and Thilikos [15]] *If G is a planar graph which has a dominating set of size k , then G has treewidth of at most $4.5^{1.5}\sqrt{k} \leq 9.55\sqrt{k}$.*

COROLLARY 5.7 PLANAR ROMAN DOMINATION *can be solved in time*

$$\mathcal{O}^*(5^{4.5^{1.5}\sqrt{k}}) = \mathcal{O}^*(2^{22.165\sqrt{k}}).$$

6 A dual version of ROMAN DOMINATION

We finally mention that the following version of a parametric dual of ROMAN is in \mathcal{FPT} by the method of kernelization, relying on [7, Proposition 4(e)]: given a graph G and a parameter k_d , is there a Roman domination function R such that $|R^{-1}(1)| + 2|R^{-1}(0)| \geq k_d$?

The definition of a dual of ROMAN DOMINATION might look a bit funny at first glance: But since ROMAN DOMINATION is a sort of weighted version of DOMINATING SET, it is not quite clear what the notion of a parametric dual should be in this case. With our definition, we have the possibly desirable property that (G, k_d) is a YES-instance of this variant of a dual of ROMAN DOMINATION iff $(G, 2|V(G)| - k_d)$ is a YES-instance of ROMAN. In other words, R is maximum for this dual version of ROMAN iff R is minimum for ROMAN.

THEOREM 6.1 *Our version of parametric dual of ROMAN DOMINATION allows for a problem kernel of size $(7/6)k_d$, measured in terms of vertices. Hence, this problem is in \mathcal{FPT} .*

Proof Note that we can easily get rid of all isolates with a first reduction rule: If x is an isolate, assign zero to x and decrease the parameter k_d by two.

As a second reduction rule, we claim that if $k_d < (6/7)|V(G)|$, then we can answer YES. Of course, this gives the claimed problem kernel.

Assume to the contrary that (G, k_d) is a NO-instance and that $k_d < (6/7)|V(G)|$. Hence, for any optimum Roman domination function R for G ,

$$|R^{-1}(1)| + 2|R^{-1}(0)| < k_d < (6/7)|V(G)|.$$

Hence, $|R^{-1}(0)| < (3/7)|V(G)|$. This is also true for any optimum Roman domination function R that also minimizes $|R^{-1}(1)|$ (as a second priority). This contradicts [7, Proposition 4(e)].

This shows that also this dual version of ROMAN DOMINATION is in \mathcal{FPT} .

□

Notice that this results parallels the situation found with DOMINATING SET [10].

7 Conclusion

This paper contains a number of technical results concerning a parameterized view on ROMAN DOMINATION. Besides these technical results, we like to communicate the following messages:

- As can be seen from the W[2] completeness section, the “Turing way” to parameterized complexity is often quite amenable and may offer advantages over the standard approach as exhibited in [11].
- Strive to obtain structural results when developing algorithms: this turned out to be very beneficial for PLANAR ROMAN DOMINATION, since the results obtained for PLANAR DOMINATING SET could be “recycled.”
- For the time analysis of the update complexity for a parent (join) node, given the children nodes in a tree decomposition, similar recurrences than for the assessment of running times of search tree algorithms can be used.

References

- [1] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. Rosamond, and U. Stege. A refined search tree techniques for DOMINATING SET on planar graphs. *Journal of Computer and System Sciences*, vol. 71 (2005), pp. 385–405.
- [2] S. Benecke. Higher order domination of graphs. Master’s thesis, Department of Applied Mathematics of the University of Stellenbosch, South Africa, 2004.
- [3] H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Automata, Languages and Programming, Proc. 15th ICALP*, volume 317 of LNCS, pp. 105–119, 1988.
- [4] M. Cesati. The Turing way to parameterized complexity. *Journal of Computer and System Sciences*, 67:654–685, 2003.
- [5] M. Cesati and M. Di Ianni. Computation models for parameterized complexity. *Mathematical Logic Quarterly*, 43:179–202, 1997.
- [6] Y. Chen and J. Flum. Machine characterization of the classes of the W-hierarchy. In *Proc. 17th CSL*, volume 2803 of LNCS, pp. 114–127. Springer, 2003.
- [7] E. J. Cockayne, P. A. Dreyer Jr., S. M. Hedetniemi, and S. T. Hedetniemi. Roman domination in graphs. *Discrete Mathematics*, 278:11–22, 2004.
- [8] E. J. Cockayne, P. J. P. Grobler, W. R. Gründlingh, J. Munganga and J. H. van Vuuren. Protection of a graph. *Utilitas Mathematica*, 67:19–32, 2005.
- [9] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. In *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of LNCS, pages 1–12. Springer, 2004.
- [10] F. Dehne, M. Fellows, H. Fernau, E. Prieto and F. Rosamond. NONBLOCKER SET: Parameterized Algorithmics for MINIMUM DOMINATING SET. In *Software Seminar SOFSEM 2006*, volume 3831 of LNCS, pp. 237–245. Springer, 2006.
- [11] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
- [12] P. A. Dreyer Jr. *Applications and Variations of Domination in Graphs*. PhD thesis, Rutgers University, New Jersey, USA, PhD Thesis, 2000.
- [13] H. Fernau. On parameterized enumeration. In *Computing and Combinatorics, Proceedings COCOON 2002*, volume 2383 of LNCS, pages 564–573. Springer, 2002.
- [14] H. Fernau. *Parameterized Algorithmics: A Graph-Theoretic Approach*. Habilitationsschrift, Universität Tübingen, Germany, 2005.
- [15] F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: branch-width and exponential speed-up. In *Proc. 14th SODA*, pp. 168–177, 2003.
- [16] W. D. Goddard, S. M. Hedetniemi and S. T. Hedetniemi. Eternal security in graphs. *J. Combin. Math. Combin. Comput.*, to appear.
- [17] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, 1998.
- [18] M. A. Henning. Defending the Roman Empire from multiple attacks. *Discrete Mathematics*, 271:101–115, 2003.
- [19] M. A. Henning and S. T. Hedetniemi. Defending the Roman Empire: a new strategy. *Discrete Mathematics*, 266:239–251, 2003.
- [20] T. Kloks. *Treewidth. Computations and Approximations*, volume 842 of LNCS. Springer, 1994.
- [21] M. Liedloff, T. Kloks, J. Liu and S.-L. Peng. Roman domination over some graphs classes. *Graph-Theoretic Concepts in Computer Science, 31st International Workshop, WG*, volume 3787 of

LNCS, pages 103–114. Springer, 2005.

- [22] A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D. S. Taylor, and P. Widmayer. Server placements, Roman domination and other dominating set variants. In *2nd IFIP International Conference on Theoretical Computer Science IFIP TCS*, pp. 280–291. Kluwer, 2002.
- [23] I. Stewart. Defend the Roman Empire! *Scientific American*, Dec.:136–139, 1999.
- [24] J. A. Telle and A. Proskurowski. Practical algorithms on partial k -trees with an application to domination-like problems. In F. Dehne et al., editors, *Algorithms and Data Structures, Proc. 3rd WADS'93*, volume 709 of *LNCS*, pp. 610–621, 1993.