Roman Domination over Some Graph Classes*

Mathieu Liedloff[1], Ton Kloks, Jiping Liu[2], and Sheng-Lung Peng[3]

[1] Université Paul Verlaine - Metz,
Laboratoire d'Informatique Théorique et Appliquée,
57045 Metz Cedex 01, France
liedloff@sciences.univ-metz.fr
[2] Department of Mathematics and Computer Science,
The university of Lethbridge,
Alberta, T1K 3M4, Canada
liu@cs.uleth.ca
[3] Department of Computer Science and Information Engineering,
National Dong Hwa University,
Hualien, Taiwan, R.O.C
lung@csie.ndhu.edu.tw

Abstract. A Roman dominating function of a graph $G = (V, E)$ is a function $f : V \to \{0, 1, 2\}$ such that every vertex x with $f(x) = 0$ is adjacent to at least one vertex y with $f(y) = 2$. The weight of a Roman dominating function is defined to be $f(V) = \sum_{x \in V} f(x)$, and the minimum weight of a Roman dominating function on a graph G is called the Roman domination number of G.

In this paper we answer an open problem mentioned in [2] by showing that the Roman domination number of an interval graph can be computed in linear time. We also show that the Roman domination number of a cograph can be computed in linear time. Besides, we show that there are polynomial time algorithms for computing the Roman domination numbers of AT-free graphs and graphs with a d-octopus.

1 Introduction

Let $G = (V, E)$ be an undirected and simple graph. A *Roman dominating function* is a function $f : V \to \{0, 1, 2\}$ such that every vertex x with $f(x) = 0$ is adjacent to at least one vertex y with $f(y) = 2$. The *weight* of a Roman dominating function is $f(V) = \sum_{x \in V} f(x)$. The minimum weight of a Roman dominating function on a graph G is called the *Roman domination number* of G and is denoted by $\gamma_{\mathsf{R}}(G)$.

Roman domination has been introduced in [2] as a new variety of the classical domination problem having both historical and mathematical interest, particularly in the field of server placements [15]. We refer to [2,6,10,11,12,16,17] for

* The second and the third authors were partially supported by NSERC of Canada. The second author was supported also by the National Science Council of Taiwan under grant NSC 93-2811-M-002-004. The first author is the corresponding author.

D. Kratsch (Ed.): WG 2005, LNCS 3787, pp. 103–114, 2005.

more background on the historical importance of the Roman domination problem and various mainly graph-theoretic results not mentioned here.

The complexity of the Roman domination problem when restricted to interval graphs was mentioned as an open problem in [2]. In this paper we show that there are linear time algorithms to compute the Roman domination number for interval graphs and cographs. We also show that there are polynomial time algorithms for computing the Roman domination numbers of AT-free graphs and graphs with a d-octopus. The paper is organized as follows. Section 2 gives some preliminaries about our problem. The results for interval graphs and cographs are presented in Sections 3 and 4, respectively. In Section 5, we present polynomial time algorithms for computing the Roman domination numbers of AT-free graphs and graphs with a d-octopus.

2 Preliminaries

Let $G = (V, E)$ be an undirected and simple graph. For a vertex x of G we denote by $N(x)$ the neighborhood of x in G and by $N[x] = N(x) \cup \{x\}$ the closed neighborhood of x. The distance $d_G(x, y)$ between two vertices x and y is the length of a shortest path joining these two vertices.

A dominating set D of a graph $G = (V, E)$ is a subset of vertices such that every vertex of $V - D$ has at least one neighbor in D. The minimum cardinality of a dominating set of G is said to be the domination number of G, and it is denoted by $\gamma(G)$. An independent set in a graph G is a subset of pairwise non-adjacent vertices.

Now let us summarize some useful facts on Roman domination [2].

Theorem 1 ([2]). $\gamma(G) \leq \gamma_{\mathsf{R}}(G) \leq 2\gamma(G)$.

Lemma 1 ([2].). *If G is a graph of order n, then $\gamma_{\mathsf{R}}(G) = \gamma(G)$ if and only if $G = \overline{K_n}$, i.e., G is an independent set with n vertices.*

Definition 1. *A* 2-packing *is a set $S \subseteq V$ such that for every pair $x, y \in S$ $N[x] \cap N[y] = \varnothing$. The maximum cardinality of a 2-packing in G is called the* 2-packing number *of G.*

Theorem 2 ([2]). *Let f be a minimum weighted Roman dominating function of a graph G without isolated vertices. Let V_i, $i = 0, 1, 2$, be the set of vertices x with $f(x) = i$. Let f be such that $|V_1|$ is the minimum. Then V_1 is a 2-packing and there is no edge between V_1 and V_2.*

Theorem 3 ([2]). *For any non-trivial connected graph G,*

$$\gamma_{\mathsf{R}}(G) = \min\{|S| + 2\gamma(G - S) \mid S \text{ is a 2-packing}\}.$$

Remark 1. A 2-packing S can serve as V_1 and a dominating set in $G - S$ as V_2. Notice that the weight of a Roman dominating function is $|V_1| + 2|V_2|$.

Definition 2. *We call* (V_1, V_2) *a* Roman pair *of a graph* G *if* (V_1, V_2) *is a solution induced by a minimum weighted Roman dominating function of the graph* G.

We refer the reader to [1,8] for definitions and properties of graph classes not given in this paper.

3 Roman Domination on Interval Graphs

Throughout this section we assume that $G = (V, E)$ is connected. Clearly, if G is disconnected then, obviously, $\gamma_{\mathsf{R}}(G)$ is the *sum* of the Roman domination numbers of its components.

Definition 3. *A graph* $G = (V, E)$ *is an* interval graph *if there exists a set* $\{I_v \mid v \in V\}$ *of intervals of the real line such that* $I_u \cap I_v \neq \varnothing$ *iff* $uv \in E$.

Both I_v and v can be used to represent the vertex v in an interval graph. Let $l(v)$ and $r(v)$ denote the values of the left and right end points of the interval $I_v = [l_v, r_v]$, respectively. A model of an interval graph is normalized if $\cup_{v \in V}\{l(v), r(v)\} = \{1, 2, \ldots, 2n\}$. In the following we assume that a normalized model of the graph is part of the input.

Our linear time algorithm to compute the Roman domination number of an interval graph uses dynamic programming and passes through the interval collection from left to right to enumerate all the potential optimum solutions (V_1, V_2).

3.1 Structure of an Optimum Solution

In this section, we examine the structure of an optimum solution.

Lemma 2. *For every interval graph there exists a Roman pair* (V_1, V_2) *such that no interval in* V_2 *is properly contained in another interval.*

Lemma 3. *If* (V_1, V_2) *is a Roman pair, then* V_2 *contains no clique of size* 3 *or more.*

Proof. Let $\{i_1, i_2, i_3\} \subseteq V_2$ be a clique of size three. By Lemma 2, there is no interval which is properly contained in another interval. Without loss of generality, we assume $l(i_1) < l(i_2) < l(i_3) < r(i_1) < r(i_2) < r(i_3)$. Then we obtain that $N[i_2] \subseteq N[i_1] \cup N[i_3]$. That is, $(V_1, V_2 \setminus \{i_2\})$ is a Roman pair of G which is a contradiction. □

Lemma 4. *If* (V_1, V_2) *is a Roman pair, then the connected components induced by* V_2 *are paths.*

Proof. By Lemma 2, each connected component induced by V_2 is a proper interval graph. Hence, it is chordal and it does not contain a claw, i.e., $K_{1,3}$. Together with Lemma 3, our lemma holds. □

We can use this last result in the following way: in order to find a set V_2 of an optimum solution, we only have to consider certain shortest paths between some pairs of vertices. Now, we characterize the set V_1 of an optimum solution.

Definition 4. *Let (V_1, V_2) be a Roman pair of an interval graph G. Intervals $\mathcal{J} \subseteq V_1$ are* consecutive *iff between the leftmost and rightmost end points of $\mathcal{J}$ there is no end point of an interval $I \in V_2$.*

Lemma 5. *There exists a Roman pair (V_1, V_2) with the property that V_1 is an independent set, and there is no subset $\mathcal{J} \subseteq V_1$ containing more than two consecutive intervals.*

Proof. By Theorem 2, we have a Roman pair (V_1, V_2) with V_1 being an independent set. Let $\{a, b, c\} \subseteq V_1$ be a set of three consecutive intervals in V_1. Suppose that $l(a) < r(a) < l(b) < r(b) < l(c) < r(c)$. Since (V_1, V_2) is of minimum weight, we have $\forall v \in N(b)$, $v \notin V_1$ and $v \notin V_2$. Consequently, if $v \in N(b)$ there must exist a $w \in N(v)$ such that $w \in V_2$. However $\{a, b, c\}$ are consecutive, therefore, we have $r(w) < l(a)$ (or resp. $r(c) < l(w)$). As a result of $v \in N(a)$ (resp. $v \in N(c)$), there exists a solution with $f(v) = 2$ and $f(a) = f(b) = 0$ (resp. $f(b) = f(c) = 0$). Consequently if we have a solution with three consecutive intervals, there exists a solution (V_1, V_2) of same weight such that V_1 contains no more than two consecutive intervals. □

3.2 Description of the Algorithm

Previous results show us how to build a potential solution (V_1, V_2). Indeed, we have seen that connected components induced by V_2 are paths and each of these paths can be preceded or followed by at most two consecutive intervals of V_1. So, our algorithm goes through the interval collection in a left-right fashion. An optimum solution, i.e, a solution whose weight is the minimum over all possible solutions, will be one of the solutions found by the algorithm with minimum value of $|V_1|+2|V_2|$. The algorithm uses dynamic programming in order to intelligently test every possible solution with respect to the structure established by previous lemmas.

For any given normalized interval graph $G = (V, E)$ of order n, the algorithm treats intervals increasingly according to their right end points. Corresponding to a right end point d ($0 \leq d \leq 2n$) of an interval, we define a sub-solution (V_1', V_2') by

1. $V_1', V_2' \subseteq \{i \in V : r(i) \leq d\}$,
2. (V_1', V_2') is a solution of minimum weight over all the solutions for the graph $G[S]$, where $S = \{v \in V : l(v) \leq d\}$, such that the interval i with $r(i) = d$ belongs to V_2'.

Clearly, at the beginning of the algorithm no intervals are yet considered and we define for $d = 0$ the sub-solution $(V_1', V_2') = (\varnothing, \varnothing)$. Then, for each step, we start with a current integer d and its corresponding sub-solution (V_1', V_2'), and we construct an *extension* (V_1'', V_2'') of (V_1', V_2') corresponding to a new d', where $d' > d$. According to previous lemmas, there are three possible cases:

1. add two intervals i_1 and i'_1 to V'_1 and one interval i_2 to V'_2 such that $(V''_1, V''_2) = (V'_1 \cup \{i_1, i'_1\}, V'_2 \cup \{i_2\})$ is a sub-solution corresponding to $d' = r(i_2)$ *(see procedure Add-intervals-first-choice)*,
2. add one interval i_1 to V'_1 and one interval i_2 to V'_2 such that $(V''_1, V''_2) = (V'_1 \cup \{i_1\}, V'_2 \cup \{i_2\})$ is a sub-solution corresponding to $d' = r(i_2)$ *(see procedure Add-intervals-second-choice)*,
3. add one interval i_2 to V'_2 such that $(V''_1, V''_2) = (V'_1, V'_2 \cup \{i_2\})$ is a sub-solution corresponding to $d' = r(i_2)$ *(see procedure Add-intervals-third-choice)*.

The first choice corresponds to adding two consecutive intervals to V'_1 and then starting a new path in V''_2. In the second case, we add one interval to V'_1 and begin a new path in V''_2. In the last case, we add only one interval to V'_2 which extends an existing path in V'_2 or begins a new path in V''_2.

Now, we provide another result which will be used in the construction of some sub-solutions.

Lemma 6. *Let d be an integer such that $1 \leq d \leq 2n$. Suppose we have a sub-solution (V'_1, V'_2) for the set of all intervals i with $l(i) < d$. Let i_1 and i'_1 be such that $r(i_1) = \min\{r(i) : l(i) > d\}$ and $r(i'_1) = \min\{r(i) : l(i) > r(i_1)\}$. Let w be such that $r(w) = \min\{r(i) : l(i) > d \ \wedge \ i \neq i_1 \ \wedge \ i \neq i'_1\}$. If $w \in N(i_1)$, then there exists an optimum solution (V''_1, V''_2) where i_1 and i'_1 are not two consecutive intervals in V''_1.*

Proof. By the construction of i_1, i'_1 and w, we have that $d < l(i_1)$, $d < l(i'_1)$, $d < l(w)$ and $r(i_1) < r(w)$. Since $w \in N(i_1)$, then $l(w) < r(i_1) < r(w)$. There are two cases.

1. $w \in N(i'_1)$. Then there exists an alternative solution with $w \in V''_2$ and $i_1, i'_1 \notin V''_1$.
2. $w \notin N(i'_1)$. Then we have $r(w) < l(i'_1)$ and there are three sub-cases:
   (a) $w \in V''_2$. Then i_1 and i'_1 are not consecutive.
   (b) There exists a $v \in N(w)$ such that $v \in V''_2$ ($w \in V''_0$). Then $l(v) < r(w) < r(v)$ and $v \in V''_2$. If both i_1 and i'_1 are in V''_1, then i_1 and i'_1 cannot be consecutive since at least one end of v is between them.
   (c) $w \in V''_1$. In this case i_1 cannot be in V''_1, thus i_1 and i'_1 cannot be consecutive. □

3.3 Preprocessing Data

In order to achieve a linear-time algorithm, we do some pre-processing so that when we run the program, the necessary data is available in constant time. In particular, the following operations must be done in constant time in order to obtain the claimed time bound.

- find i, j, k such that $r(i) = \min\{r(v) : l(v) > d\}$, $r(j) = \min\{r(v) : l(v) > d \ \wedge \ v \neq i\}$ and $r(k) = \min\{r(v) : l(v) > d \ \wedge \ v \neq i \ \wedge \ v \neq j\}$ for a fixed d,
- find i such that $r(i) = \max\{r(v) : v \in N[x]\}$ for a fixed x,
- check whether $N[x] \cap N[y] \neq \varnothing$ for two intervals x and y such that $r(x) < r(y)$ (for this operation we only have to find i such that $r(i) = \max\{r(v) : v \in N[x]\}$ and then check whether $i \in N[y]$).

Sort Intervals According to Their Right End Points (SIRE). The collection of n intervals is given in a normalized interval model. We sort the intervals in an array D of size $2n$ such that $D[i] = \begin{cases} j & \text{if } \exists j \text{ s.t. } r(j) = i, \\ \texttt{NIL} & \text{otherwise.} \end{cases}$ in time $O(n)$ using bucket sort.

Find Three Intervals with Lowest Right End Points (ILRE). Now, we use the array D to build another 2-dimensional array $MinR$ which contains for each value $d \in \{0, 1, \ldots, 2n\}$ the first, second, and third interval whose right end points are the first, second, and third lowest, respectively, and such that their left end points are greater than d.

Find Intervals with Greatest Right End Points (IGRE). Finally, we calculate for each interval $i \in \{1, \ldots, n\}$ its neighbor which has the greatest right end point, or the interval i if there is no such a neighbor, in an array $MaxR$.

The three procedures SIRE, ILRE and IGRE have been shown in detail in [13], and each takes $O(n)$ time.

3.4 A Linear-Time Algorithm

Using the structure of an optimum solution described by previous lemmas of this section and some results stated in section 2 (in particular Theorem 2), we are ready to present a linear-time algorithm for solving the Roman domination problem on interval graphs. An optimum solution can be easily constructed by standard techniques.

Procedure Add-intervals-first-choice(d)
Data: An integer d such that a corresponding sub-solution (V_1', V_2') has already been computed.
Result: An extension of the sub-solution (V_1', V_2') constructed using the first case (add two intervals to V_1' and one interval to V_2').

$i_1 \leftarrow MinR[d][1]$
if $i_1 \neq$ *NIL* **then**
  $i_1' \leftarrow MinR[r(i_1)][1]$
  if $i_1' \neq$ *NIL* **then**
    if $MaxR[i_1]$ *does not intersect* i_1' **then**
      $w \leftarrow MinR[d][2]$
      if $w = i_1'$ **then** $w \leftarrow MinR[d][3]$
      if $w \neq$ *NIL* **then**
        if i_1 *does not intersect* w **then**
          $i_2 \leftarrow MaxR[w]$
          if i_1 *does not intersect* i_2 ***and*** i_1' *does not intersect* i_2 **then**
            $Weight[r(i_2)] \leftarrow \min\{Weight[r(i_2)], Weight[d] + 4\}$
      else $Weight[2n] \leftarrow \min\{Weight[2n], Weight[d] + 2\}$

Procedure Add-intervals-second-choice(d)
Data: An integer d such that a corresponding sub-solution (V_1', V_2') has already been computed.
Result: An extension of the sub-solution (V_1', V_2') constructed using the second case (add one interval to V_1' and one interval to V_2').

$i_1 \leftarrow MinR[d][1]$
if $i_1 \neq$ `NIL` **then**
  $w \leftarrow MinR[d][2]$
  if $w \neq$ `NIL` **then**
    $i_2 \leftarrow MaxR[w]$
    if i_1 *does not intersect* i_2 **then**
      $Weight[r(i_2)] \leftarrow \min\{Weight[r(i_2)], Weight[d] + 3\}$
  else $Weight[2n] \leftarrow \min\{Weight[2n], Weight[d] + 1\}$

Procedure Add-intervals-third-choice(d)
Data: An integer d such that a corresponding sub-solution (V_1', V_2') has already been computed.
Result: An extension of the sub-solution (V_1', V_2') constructed using the third case (add one interval to V_2').

$w \leftarrow MinR[d][1]$
if $w \neq$ `NIL` **then**
  $i_2 \leftarrow MaxR[w]$
  $Weight[r(i_2)] \leftarrow \min\{Weight[r(i_2)], Weight[d] + 2\}$
else $Weight[2n] \leftarrow \min\{Weight[2n], Weight[d]\}$

Algorithm Roman-Dom(normalized interval model of a graph G ; $\gamma_{\mathsf{R}}(G)$)
Data: An interval graph represented by a normalized model.
Result: The Roman domination number γ_{R} of the input interval graph.
Construct the data structures D, $MinR$ and $MaxR$

for $i = 1$ *to* $2n$ **do** $Weight[i] \leftarrow 2n$
$Weight[0] \leftarrow 0$
`Add-intervals-first-choice(0)`
`Add-intervals-second-choice(0)`
`Add-intervals-third-choice(0)`
for $i = 1$ *to* $2n$ **do**
  if $D[i] \neq$ `NIL` ***and*** $Weight[r(D[i])] \neq 2n$ **then**
    `Add-intervals-first-choice(`$r(D[i])$`)`
    `Add-intervals-second-choice(`$r(D[i])$`)`
    `Add-intervals-third-choice(`$r(D[i])$`)`
return $\gamma_{\mathsf{R}}(G) = Weight[2n]$

Theorem 4. *The Roman domination problem can be solved in $O(n)$ time on any interval graph with a normalized interval model.*

Proof. The correctness of the algorithm follows from the lemmas stated in Subsections 3.1 and 3.2

We note that it takes linear time to construct D, $MinR$ and $MaxR$, and it takes constant time to process each of the procedures **Add-intervals-first-**

choice, **Add-intervals-second-choice**, and **Add-intervals-third-choice**. The complexity of the algorithm **Roman-Dom** is dominated by the second **for** loop. Therefore, the complexity of the algorithm is $O(n)$. □

4 Roman Domination on Cographs

In this section we describe an algorithm to compute the Roman domination number of a cograph G. We may assume that G is connected, since otherwise $\gamma_{\mathsf{R}}(G)$ equals the sum of the Roman domination numbers of its components.

If G is connected then G is the *join* of two graphs G_1 and G_2. Clearly, *any* 2-packing of G consists of at most one vertex since G is P_4-free. By Theorem 3 the Roman domination number of G can be computed by taking the minimum over all vertices x of $2\gamma(G-x)+1$ and $2\gamma(G)$. It is well-known that the domination number of a cograph can be computed in linear time. Thus, we can compute the Roman domination number of G in $O(n(m+n))$ time, where n and m are the numbers of the vertices and edges of G respectively. However, we can obtain a linear time algorithm by using the structure of cotree.

It is well-known that any cograph G can be represented by a cotree $\mathcal{T}$ [9]. In $\mathcal{T}$, each leaf represents a vertex of G and each internal node represents either a **join** or a **union**. For any two vertices u and v, if (u,v) is an edge of G, then the lowest common ancestor of u and v in $\mathcal{T}$ is a **join** node. Since G is connected, the root of $\mathcal{T}$ is a **join** node. We may assume that $\mathcal{T}$ is a binary tree. For a node v, let $\mathcal{T}_v$ denote the subtree of $\mathcal{T}$ rooted at v. Let G_v denote the subgraph defined by $\mathcal{T}_v$. Now, our algorithm is as follows.

For a cograph G, we traverse its corresponding cotree $\mathcal{T}$ from leaves to the root. Let $(V_1(G_v), V_2(G_v))$ be a Roman pair of G_v. Initially, every leaf w is in $V_1(G_w)$ and $V_2(G_w)$ is empty, i.e., $\gamma_{\mathsf{R}}(G_w)=1$. Now let us consider an internal node u in $\mathcal{T}$, let l (respectively, r) be its left (respectively, right) child. That is, G_u is the resulting cograph by applying union or join operation on G_l and G_r. If u is a **union** node, then $(V_1(G_u), V_2(G_u)) = (V_1(G_l)\cup V_1(G_r), V_2(G_l)\cup V_2(G_r))$ is a Roman pair of G_u. If u is a **join** node, we do the following. Without loss of generality, let $\gamma_{\mathsf{R}}(G_l) \le \gamma_{\mathsf{R}}(G_r)$.

1. $\gamma_{\mathsf{R}}(G_l) = \gamma_{\mathsf{R}}(G_r)$. If at least one of $V_2(G_l)$ and $V_2(G_r)$ is not empty, say $V_2(G_l)\neq\emptyset$, then set $V_1(G_r)=V_2(G_r)=\emptyset$. We do this because every vertex in G_r is dominated by a vertex $v\in V_2(G_l)$.
   If both $V_2(G_l)$ and $V_2(G_r)$ are empty, then we move any vertex $v\in V_1(G_l)$ to $V_2(G_l)$. We then set $V_1(G_r)=V_2(G_r)=\emptyset$ for the same reason.
2. $\gamma_{\mathsf{R}}(G_l) < \gamma_{\mathsf{R}}(G_r)$. If $V_2(G_l)=\emptyset$, again we move a vertex $v\in V_1(G_l)$ to $V_2(G_l)$. Since every vertex in G_r is dominated by v, we set $V_1(G_r)=V_2(G_r)=\emptyset$.
   If $V_2(G_l)\neq\emptyset$, then we set $V_1(G_r)=V_2(G_r)=\emptyset$ for the same reason.

In any one of above cases, if $2|V_2(G_l)|+|V_1(G_l)|>4$, then (i) keep only one vertex in $V_2(G_l)$, (ii) set $V_1(G_l)=\emptyset$, and (iii) arbitrarily select a vertex in G_r and add it to $V_2(G_r)$. Finally, let $V_i(G_u)=V_i(G_l)\cup V_i(G_r)$ for $i=1,2$. It is not

hard to see that $\gamma_{\mathsf{R}}(G) \leq 4$ for any connected cograph G. We have the following theorem.

Theorem 5. *The Roman domination number of a cograph can be computed in linear time.*

Proof. For the correctness, we show it by induction on the height of $\mathcal{T}$. In the base case that we consider the height equal to 0. Since every vertex w is an isolated vertex, $\gamma_{\mathsf{R}}(G_w) = 1$. Thus, $(\{w\}, \emptyset)$ is the Roman pair of G_w. Assume that for any node v in $\mathcal{T}$ with height equal to h, we can compute a Roman pair $(V_1(G_v), V_2(G_v))$ for G_v. Now, consider a node u with height $h+1$. Let l and r be its left and right children in $\mathcal{T}$, respectively. If u is a **union** node, it is easy to check that $(V_1(G_l) \cup V_1(G_r), V_2(G_l) \cup V_2(G_r))$ is a Roman pair of G_u. We now consider the case that u is a **join** node. Without loss of generality, we assume that $\gamma_{\mathsf{R}}(G_l) \leq \gamma_{\mathsf{R}}(G_r)$. By the definition, every vertex in G_r is adjacent to any vertex of G_l. If $V_2(G_l)$ is not empty, then every vertex is dominated by a vertex in $V_2(G_l)$. Thus $(V_1(G_l), V_2(G_l))$ can Roman dominate G_u. If $V_2(G_l)$ is empty, we can promote a vertex in $V_1(G_l)$ to $V_2(G_l)$ such that it can dominate G_r. Since $\gamma_{\mathsf{R}}(G_l) \leq \gamma_{\mathsf{R}}(G_r)$, we can obtain a better solution by doing so. However, it will increase the weight of the Roman dominating function. If $|V_1(G_l)| + 2|V_2(G_l)| \leq 4$, then $(V_1(G_l), V_2(G_l))$ is a Roman pair of G_u. If $|V_1(G_l)| + 2|V_2(G_l)| > 4$, we select a vertex v_l from $V_2(G_l)$ and arbitrarily select a vertex v_r from G_r. Since v_l dominates G_r and v_r dominates G_l, $(\emptyset, \{v_l, v_r\})$ is a Roman pair of G_u. This show the correctness of our algorithm.

For the time complexity, we implement each dominating set using a linked list with *front* and *tail* pointers. Thus the Roman pair of a **union** node can be computed in constant time. For a **join** node, it costs constant time to empty a set. For the other operations, at most constant number of vertices are updated. Thus, the overall time complexity is linear. □

Remark 2. In [2] a graph G is called *Roman* if $\gamma_{\mathsf{R}}(G) = 2\gamma(G)$. It is proved that a graph G is Roman if and only if $\gamma(G) \leq \gamma(G - S) + \frac{|S|}{2}$ for *every* 2-packing S in G. It follows that a connected cograph G is Roman if and only if $\gamma(G) = \gamma(G - x)$ for *every* vertex x. Since, in [2] it is posed as an open problem to determine Roman graphs other than trees[4], it would be of interest to know which cographs satisfy this equality. Notice that a large subclass of Roman cographs can be constructed as follows: Take *any* cograph G and construct a graph H by replacing every vertex of G by a *true twin*. It is easy to check that H is a cograph[5], and furthermore for every vertex x in H, $\gamma(H) = \gamma(H - x)$.

5 Roman Domination on AT-Free Graphs and Graphs with a d-Octopus

In this section we study the Roman domination problem on AT-free graphs and graphs with d-octopus. Our approaches are based on algorithms for the

[4] A constructive characterization of Roman trees is given in [10].
[5] Any induced P_4 would lead to an induced P_4 in G.

domination problem in [7,14]. First we provide some preliminaries on AT-free graphs and d-octopus.

Definition 5. *Three vertices x, y and z of a graph $G = (V, E)$ form an* asteroidal triple, AT *for short, if for any two of the three vertices there is a path between them that avoids the neighborhood of the third. A graph is said to be* AT-free *if it does not contain an* AT.

Definition 6. *A pair of vertices x and y is a* dominating pair *of a graph G, if the vertex set of any path between x and y in G is a dominating set in G.*

Theorem 6 ([4]). *Any connected* AT*-free graph has a dominating pair.*

Definition 7. *A path $P = (x = x_0, x_1, \ldots, x_d = y)$ is a* dominating shortest path, DSP *for short, of a graph $G = (V, E)$ if*

1. *P is a shortest path between x and y in G,*
2. *$\{x_0, x_1, \ldots, x_d\}$ is a dominating set of G.*

Corollary 1 ([14]). *Every connected* AT*-free graph has a* DSP.

Definition 8. *A d-*octopus *O of a graph $G = (V, E)$ is a subgraph of G such that*

1. *the vertices of O is a dominating set of G,*
2. *there are vertices $r, v_1, v_2, \ldots, v_d$ of G, and for each $i \in \{1, \ldots, d\}$ there is a shortest path P_i from r to v_i in G such that O is the union of the paths $P_1, P_2, \ldots, P_d$.*

We call the common end point r of the d shortest paths the root *of the d-octopus O. Note that the paths need not to be disjoint.*

Remark 3. A graph with a DSP is a 1-octopus graph.

The following results are Roman domination versions of Lemma 33 in [7] and Theorem 4 in [14] "replacing D by V_2".

Theorem 7. *Let $G = (V, E)$ be a graph with a d-octopus with root x. Let H_0, H_1, ..., H_l be the levels of the* BFS*-tree with the root x. Then G has a Roman pair (V_1, V_2) such that:*

$$\bigwedge_{i\in\{0,1,\ldots,l\}} \bigwedge_{j\in\{0,1,\ldots,l-i\}} \left| V_2 \cap \bigcup_{s=i}^{i+j} H_s \right| \leq (j+5)d - 1. \tag{1}$$

Theorem 8. *Let $G = (V, E)$ be a connected* AT-free *graph. There is a vertex x which can be determined in linear time such that if H_0, H_1, ..., H_l are the levels of the* BFS*-tree with the root x, then G has a Roman pair (V_1, V_2) such that:*

$$\bigwedge_{i\in\{0,1,\ldots,l\}} \bigwedge_{j\in\{0,1,\ldots,l-i\}} \left| V_2 \cap \bigcup_{s=i}^{i+j} H_s \right| \leq j + 3. \tag{2}$$

A Polynomial Time Algorithm:

Our algorithm uses dynamic programming to compute a Roman pair through the levels of a BFS-tree. A subsolution computed during the execution of the algorithm is a set $S \subseteq \bigcup_{j=0}^{i-1} H_j$ chosen up to a fixed level $i-1 \in \{1, 2, \ldots, l-1\}$. Information of any subsolution S that we must store during the execution are the vertices that belong to the last two current levels (i.e, $S \cap (H_{i-2} \cup H_{i-1})$). Consequently, the number of vertices from V_2 that a Roman pair (V_1, V_2) might have in any three consecutive BFS-levels is important for the complexity of the algorithm. The previous theorems guarantee that this number is at most 5 for connected AT-free graphs and at most $7d-1$ for graphs with a d-octopus.

The algorithm $rp_k(G)$, where k is a fixed positive integer, computes a Roman pair of the given connected graph G. If G has a vertex x and a Roman pair (V_1, V_2) such that at most k vertices of V_2 belong to any three consecutive levels of the BFS-tree which has x as a root, then $rp_k(G)$ outputs a Roman pair for G.

Algorithm $rp_k(G)$
  $D \leftarrow V$
  $val(D) \leftarrow |V|$ `/* initialization: every vertex of` V `is in` V_1`, this is a trivial Roman dominating set */`
  forall $x \in V$ **do**
    Compute the BFS-level of vertex x
    $H_0 = \{x\}$, $H_1 = N(x)$, ..., $H_l = \{u \in V : d_G(x,u) = l\}$
    $i \leftarrow 1$
    Initialize the queue A_1 to contain an ordered triple $(S, S, val(S))$ for all nonempty subsets S of $N[x]$ satisfying $|S| \leq k$ with $val(S) \leftarrow 2|S|$
    Add to the queue A_1 the ordered triple $(\varnothing, \varnothing, 1)$
    while $A_i \neq \varnothing$ ***and*** $i < l$ **do**
      $i \leftarrow i+1$
      forall *triples* $(S, S', val(S'))$ *in the queue* A_{i-1} **do**
        forall $U \subseteq H_i$ *with* $|S \cup U| \leq k$ **do**
          $R \leftarrow (S \cup U) \backslash H_{i-2}$
          $R' \leftarrow S' \cup U$
          $val(R') \leftarrow val(S') + 2|U| + |H_{i-1} \backslash N[S \cup U]|$
          if *there is no triple in* A_i *with first entry* R **then**
            Insert $(R, R', val(R'))$ in the queue A_i
          if *there is a triple* $(P, P', val(P'))$ *in* A_i *such that* $P = R$ ***and*** $val(R') < val(P')$ **then**
            Replace $(P, P', val(P'))$ in A_i by $(R, R', val(R'))$
    Among all triples $(S, S', val(S'))$ in the queue A_l, determine one with minimum value $v = val(S') + |H_l \backslash N[S]|$, say $(B, B', val(B'))$
    if $v < val(D)$ **then** $D \leftarrow B'$; $val(D) \leftarrow v$
  return $(V_1, V_2) = (V \backslash N[D], D)$

Theorem 9. *Algorithm* $rp_k(G)$ *computes a Roman pair of the given connected graph* G *in time* $O(n^{k+2})$ *if* G *has a Roman pair* (V_1, V_2) *and a vertex* $x \in V$ *such that at most* k *vertices of* V_2 *belong to any three consecutive* BFS*-levels of* x.

Proof. The correctness can be seen easily and the analysis of the running time is the same as the Theorem 5 in [14]. □

Theorem 10. *There is an $O(n^{7d+1})$-time algorithm to compute Roman pairs for graphs with a d-octopus. In particular, there is an $O(n^7)$-time algorithm to calculate Roman pairs for graphs having a* DSP *and there is an $O(n^6)$-time algorithm to compute Roman pairs for* AT*-free graphs.*

Proof. By combining Theorems 7 and 9 and using the results in [3,5,14] we obtain the theorem (see [13] for more details). □

Acknowledgement. We would like to thank Dieter Kratsch for his helpful comments and advices.

References

1. Brändstadt, A., V. Le, and J. P. Spinrad, *Graph classes: A survey*, SIAM Monogr. Discrete Math. Appl., Philadelphia, 1999.
2. Cockayne, E. J., P. A. Jr. Dreyer, S. M. Hedetniemi, and S. T. Hedetniemi, Roman domination in graphs, *Discrete Math.* **278**, (2004), pp. 11–22.
3. Corneil, D. G., S. Olariu, and L. Stewart, Linear time algorithms for dominating pairs in asteroidal triple-free graphs, *SIAM J. Comput.* **28**, (1999), pp. 1284–1297.
4. Corneil, D. G., S. Olariu, and L. Stewart, Asteroidal triple-free graphs, *SIAM J. Discrete Math.* **10**, (1997), pp. 399–430.
5. Deogun, J. S. and D. Kratsch, Diametral path graphs, *Proceedings of WG'95*, *LNCS* **1017**, (1995), pp. 344–357.
6. Fernau, H., Roman domination: a parameterized perspective, Manuscript.
7. Fomin, F. V., D. Kratsch, and H. Müller, Algorithms for graphs with small octopus, *Discrete Appl. Math.* **134**, (2004), pp. 105–128.
8. Golumbic, M. C., *Algorithmic graph theory and perfect graphs*, Academic Press, New York, 1980.
9. Habib, M. and C. Paul, A simple linear time algorithm for cograph recognition, *Discrete Appl. Math.* **145**, (2005), pp. 183–197.
10. Henning, M. A., A characterization of Roman trees, *Discuss. Math. Graph Theory* **22**, (2002), pp. 225–234.
11. Henning, M, A., Defending the Roman empire from multiple attacks, *Discrete Math.* **271**, (2003), pp. 101–115.
12. Henning, M. A. and S. T. Hedetniemi, Defending the Roman empire–A new strategy, *Discrete Math.* **266**, (2003), pp. 239–251.
13. Kloks, T., M. Liedloff, J. Liu and S. L. Peng, Roman domination in some special classes of graphs, Technical Report TR-MA-04-01, Nov. 2004, University of Lethbridge, Alberta, Canada.
14. Kratsch, D., Domination and total domination on asteroidal triple-free graphs, *Discrete Appl. Math.* **99**, (2000), pp. 111–123.
15. Pagourtzis, A., P. Penna, K. Schlude, K. Steinhfel, D. S. Taylor and P. Widmayer, Server placements, Roman domination and other dominating set variants, *IFIP TCS Conference Proceedings* **271**, (2002), pp. 280–291.
16. ReVelle, C. S. and K. E. Rosing, Defenders imperium Romanum: A classical problem in military strategy, *Amer. Math. Monthly* **107**, (2000), pp. 585–594.
17. Stewart, I., Defend the Roman empire! *Sci. Amer.* **281**, (1999), pp. 136–139.