Roman domination in some special classes of graphs

by

M. Liedloff, T. Kloks, J. Liu, and S. H. Peng

Report TR-MA-04-01
November 2004.
Roman domination in some special classes of graphs

Mathieu Liedloff\(^1\), Ton Kloks, Jiping Liu\(^2\), and Sheng-Lung Peng\(^3\)

\(^1\) Université de Metz
LITA, 57045 Metz Cedex 01
France
\texttt{liedloff@sciences.univ-metz.fr}

\(^2\) Department of Mathematics and Computer Science
The university of Lethbridge
Alberta, T1K 3M4, Canada
\texttt{liu@cs.uleth.ca}

\(^3\) Department of Computer Science and Information Engineering
National Dong Hwa University
Hualien, Taiwan, R.O.C.
\texttt{lung@csie.ndhu.edu.tw}

Abstract. A Roman dominating function of a graph \(G = (V, E)\) is a function \(f : V \rightarrow \{0, 1, 2\}\) 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 [3] 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 \(f : V \rightarrow \{0, 1, 2\}\) 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 \(\gamma_R(G)\).

Roman domination has been introduced in [3] as a new variety of the classical domination problem having both historical and mathematical interest. We refer to [3–5, 9, 14–16, 20, 21] for more background on the historical importance of the Roman domination problem and various mainly graph-theoretic results not mentioned here.
It is mentioned in [3] that the Roman domination number of a tree can be computed in linear time. It is also stated in [3] that the Roman domination problem is NP-complete when restricted to split graphs, bipartite graphs, and planar graphs. The complexity of the Roman domination problem when restricted to interval graphs was mentioned as an open problem in [3]. 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 $d$-octopus. The remaining of the paper is organized as follows. Section 2 gives some preliminaries about our problem. The results for interval graphs and cographs are presented in Section 3 and 4, respectively. In Section 5, we show there are polynomial time algorithms for computing the Roman domination numbers of AT-free graphs and $d$-octopus. Some proofs are shown in appendix. Finally, we give a conclusion in the final section.

2 Preliminaries

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

Recall that a dominating set $D$ of graph $G = (V, E)$ is a subset of vertices such that every vertex of $V - D$ has at least one neighbour 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 $\gamma(G)$.

By $\alpha(G)$ we denote the independence number of a graph $G$, i.e., the maximum cardinality of a set of pairwise non-adjacent vertices.

Now let us summarize some useful facts on Roman domination.

**Theorem 1 (see [3]).** $\gamma(G) \leq \gamma_R(G) \leq 2\gamma(G)$.

**Lemma 1 (see [3]).** If $G$ is a graph of order $n$, then $\gamma_R(G) = \gamma(G)$ if and only if $G = K_n$, 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 $x, y \in S$ $N[x] \cap N[y] = \emptyset$. The maximum cardinality of a 2-packing in $G$ is called the 2-packing number of $G$ and denoted by $P_2(G)$.

**Remark 1.** $P_2(G) = \alpha(G^2)$.

**Theorem 2 (see [3]).** Let $f$ be a minimum weight Roman dominating function of a graph $G$ without isolated vertices.\(^4\) 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

1. $V_1$ is a 2-packing and

\(^4\) A fortiori, $G$ is a graph with at least two vertices.
2. there is no edge between $V_1$ and $V_2$.

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

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

**Remark 2.** 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 the graph $G$ if $(V_1, V_2)$ is a solution induced by a minimum weight Roman dominating function of the graph $G$.

**Remark 3.** Of course, if we know the set $V_2$ induced by a minimum weight Roman dominating function of a graph $G = (V, E)$, we can deduce $V_1 = V - N[V_2]$.

**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 $I_v \cap I_u \neq \emptyset$ iff $vu \in E$.

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 point of the interval $I_v = [l_v, r_v]$, respectively. An interval graph is given by a normalized model if $\cup_{v \in V} \{l(v), r(v)\} = \{1, 2, \ldots, 2n\}$.

We refer the reader to [1, 11, 17] for definitions and properties of graph classes not given in this paper.

### 3 Roman domination in interval graphs

Throughout this section we assume that $G = (V, E)$ is connected. Clearly, if $G$ is disconnected then, obviously, $\gamma_R(G)$ is the sum of the Roman domination numbers of its components. We also assume that the interval graph $G$ is given with a normalized model.

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.** If $(V_1, V_2)$ is a Roman pair, then in $V_2$ there is no interval which is properly contained by another interval.

**Proof.** Assume that there is an interval $i \in V_2$ which is properly contained in $j \in V_2$. By the definition, $N[i] \subseteq N[j]$. Then $(V_1, V_2 \setminus \{i\})$ is a Roman pair which contradicts that $(V_1, V_2)$ is a Roman pair. \(\square\)
Lemma 3. If $(V_1, V_2)$ is a Roman pair, then $V_2$ contains no clique of size 3 or more.

Proof. Let $\{i_1, i_2, i_3\} \subseteq V_2$ induce a clique of size three. By Lemma 2, there is no interval which is properly contained by 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 $N[i_2] \subseteq N[i_1] \cup N[i_3]$.

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 does not contain a claw, i.e., $K_{1,3}$. Together with Lemma 3, our lemma holds.

Definition 4. Let $(V_1, V_2)$ be a Roman pair of an interval graph $G$. Intervals $J \subseteq V_1$ are consecutive iff between the leftmost and rightmost end point of $J$ 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 $J \subseteq V_1$ 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 $\{a, b, c\} \subseteq V_1$ a set of three consecutive intervals in $V_1$. Suppose that $l(a) < r(a) < l(b) < r(b) < l(c) < r(c)$. As $(V_1, V_2)$ 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 $r(w) < l(a)$ (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 this paths can be preceded and 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 over all solutions found by the algorithm and have the minimum value $|V_1| + 2|V_2|$. 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 $G$ of order $n$, the algorithm treats intervals increasingly according to their right end points. Let $d$ be an integer variable which takes on the values of the right end points of the intervals. Then $1 \leq d \leq 2n$. If $d = 1$, a partial solution is $(V_1', V_2')$, where $V_1'$ and $V_2'$ are empty sets. For each step, we start with a current integer $d$ and a sub-solution $(V_1'', V_2'')$ of the set of intervals $j$ with $l(j) < d$ such that the interval $i$ with $r(i) = d$ is in $V_2'$ (the value $d$ and the value $|V_1'| + 2|V_2'|$ give a sub-solution of the sub-problem defined by all the intervals $j$ with $l(j) < d$). We use $(V_1'', V_2'')$ to denote an extension of $(V_1', V_2')$, then there are the following 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 partial solution with $d' = r(i_2)$,
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 partial solution with $d' = r(i_2)$,
3. add one interval $i_2$ to $V_2'$ such that $(V_1'', V_2'') = (V_1', V_2' \cup \{i_2\})$ is a partial solution with $d' = r(i_2)$.

The first choice corresponds to adding two consecutive intervals to $V_1'$ and the beginning of a new path in $V_2''$. In the second case, we add one interval to $V_1'$ and begins 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 more results which will be used in the construction of some sub-modules.

**Lemma 6.** 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 $r(w) = \min\{r(i) : l(i) > d \land i \neq i_1 \land i \neq i_1'\}$. 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 alternate 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) $w \in V_1''$. Then $i_1$ and $i_1'$ are not consecutive.
   (b) There exists a $v \in N(w)$ such that $v \in V_2''$ ($w \in V_1''$). 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. 

\[\Box\]
Lemma 7. Let \( i \) be the interval such that \( r(i) \) is the smallest among all intervals and for which there is a Roman pair \((V_1, V_2)\) with \( i \in V_1 \). Then either there is no interval \( j \in V_1 \) consecutive to \( i \), or there is a Roman pair \((V_1', V_2')\) such that \( i \not\in V_1' \cup V_2' \).

Proof. Suppose that there is a \( j \in V_1 \) such that \( i \) and \( j \) are consecutive. Let \( k \) be an interval in \( V_2 \) such that \( l(k) \) is the smallest among the intervals in \( V_2 \). Since \( G \) is connected, there must be an interval \( p \in V_0 \) such that \( p \) is adjacent to \( k, i, j \). Let \( V_2' = V_2 \cup \{p\} \) and \( V_1' = V_1 - \{i, j\} \). Then \((V_1', V_2')\) is a Roman pair. This completes the proof. \( \Box \)

3.3 Preprocessing data

In order to obtain a linear-time algorithm, we must pre-calculate some data before the algorithm so that when we run the algorithm, the data will be obtained in a constant time. In particular, the following operations must be done in a constant time in order to preserve the time spent by the algorithm:

- 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 \( r(k) = \min\{r(v) : l(v) > d \wedge v \neq i \text{ and } v \neq j\} \) for a fixed \( d \),
- find \( i \) such that \( r(i) = \max\{r(v) : v \in N[x]\} \) for a fixed \( x \),
- check whether \( N[x] \cap N[y] \neq \emptyset \) 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 \( i \in N[y]\)).

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>1</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>11</td>
</tr>
</tbody>
</table>

Fig. 1. an interval collection given by their normalized model

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 time \( O(n) \) using bucket sort.

1: for \( i = 1 \) to \( 2n \) do
2: \( D[i] \leftarrow \text{NIL} \)
3: for \( i = 1 \) to \( n \) do
Example 1. For the collection of intervals as shown in Fig. 2, we obtain the following array $D$:

```
NIL NIL NIL 3 NIL 4 2 NIL NIL 1 6 5
```

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

```java
1: for $i = 0$ to $2n$ do
2: for $j = 1$ to $3$ do
3: $\text{MinR}[i][j] \leftarrow \text{NIL}$
4: $\text{indexMinR} \leftarrow 0$
5: for $i = 1$ to $2n$ do
6: if $D[i] \neq \text{NIL}$ then
7: while $l(D[i]) > \text{indexMinR}$ do
8: $\text{MinR}[\text{indexMinR}][1] \leftarrow D[i]$
9: $\text{indexMinR} \leftarrow \text{indexMinR} + 1$
10: for $j = 2$ to $3$ do
11: $\text{prev} \leftarrow 0$
12: $\text{indexMinR} \leftarrow 0$
13: for $i = r(\text{MinR}[0][j-1]) + 1$ to $2n$ do
14: if $D[i] \neq \text{NIL}$ then
15: if $\text{prev} \neq \text{MinR}[\text{indexMinR}][j-1]$ and $\text{prev} \neq 0$ then
16: $\text{prev} \leftarrow \text{MinR}[\text{indexMinR}][j-1]$
17: while $l(D[i]) > \text{indexMinR}$ and ($\text{MinR}[\text{indexMinR}][j-1] = \text{prev}$ or $\text{prev} = 0$) do
18: $\text{MinR}[\text{indexMinR}][j] \leftarrow D[i]$
19: $\text{prev} \leftarrow \text{MinR}[\text{indexMinR}][j-1]$
20: $\text{indexMinR} \leftarrow \text{indexMinR} + 1$
```

Example 2. For the previous collection, we obtain the following array $\text{MinR}$:

```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st interval</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td></td>
</tr>
<tr>
<td>2nd interval</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td></td>
</tr>
<tr>
<td>3rd interval</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td>NIL</td>
<td></td>
</tr>
</tbody>
</table>
```

Find intervals with greatest right end points (IGRE). Finally, we calculate for each interval $i \in \{1, \ldots, n\}$ its neighbour which has the greatest right end point, or the interval $i$ if there is no such a neighbour.
1: \text{$\text{indexMaxR} \leftarrow 1$} \{first, we order the intervals decreasingly according to their right end point\}

2: for $i = 2n$ to 1 do

3: \text{if $D[i] \neq \text{NIL}$ then}

4: $D'[\text{indexMaxR}] \leftarrow D[i]$

5: $\text{indexMaxR} \leftarrow \text{indexMaxR} + 1$

\{then, we construct the array which contain $u$ such that $r(u) = \max\{r(v) : v \in N[i]\}$ for all $i$\}

6: $\text{indexMaxR} \leftarrow 1$

7: for $i = 2n$ to 1 do

8: \text{if $D[i] \neq \text{NIL}$ then}

9: \text{while $\text{indexMaxR} \leq n$ and $l(D'[\text{indexMaxR}]) \leq r(D[i])$ and $l(D[i]) \leq r(D'[\text{indexMaxR}])$ do}

10: \{i.e $D'[\text{indexMaxR}]$ is a neighbour of $D[i]$\}

11: $\text{MaxR}[D'[\text{indexMaxR}]] \leftarrow D[i]$

12: $\text{indexMaxR} \leftarrow \text{indexMaxR} + 1$

\textbf{Example 3.} For the previous collection, we obtain the following array $\text{MaxR}$:

\begin{center}
\begin{tabular}{ccccccc}
interval 1 & interval 2 & interval 3 & interval 4 & interval 5 & interval 6 \\
5 & 1 & 2 & 1 & 5 & 5
\end{tabular}
\end{center}

\subsection*{3.4 The linear-time algorithm}

Using the structure of an optimum solution described by previous lemmas of this section and some results stated in the introduction of this paper (in particular Theorem 2 on page 2), we are ready to present a linear-time algorithm for solving the Roman domination problem on interval graphs. It’s easy to see that a solution can be constructed by a standard backtracking technique.

\textbf{Add-intervals-first-choice($d$)}

1: $i_1 \leftarrow \text{MinR}[d][1]$

2: \text{if $i_1 \neq \text{NIL}$ then}

3: $i'_1 \leftarrow \text{MinR}[r(i_1)][1]$

4: \text{if $i'_1 \neq \text{NIL}$ then}

5: \text{if $\text{MaxR}[i_1]$ does not intersect $i'_1$ then}

6: $w \leftarrow \text{MinR}[d][2]$

7: \text{if $w = i'_1$ then}

8: $w \leftarrow \text{MinR}[d][3]$

9: \text{if $w \neq \text{NIL}$ then}

10: \text{if $i_1$ does not intersect $w$ then}

11: $i_2 \leftarrow \text{MaxR}[w]$

12: \text{if $i_1$ does not intersect $i_2$ and $i'_1$ does not intersect $i_2$ then}

13: $\text{Weight}[r(i_2)] \leftarrow \min\{\text{Weight}[r(i_2)], \text{Weight}[d] + 4\}$

14: \text{else}

15: $\text{Weight}[2n] \leftarrow \min\{\text{Weight}[2n], \text{Weight}[d] + 2\}$
Add-intervals-second-choice(d)
1: \text{i}_1 \leftarrow \text{MinR}[d][1]
2: \text{if} \ i_1 \neq \text{NIL} \text{ then}
3: \ w \leftarrow \text{MinR}[d][2]
4: \text{if} \ w \neq \text{NIL} \text{ then}
5: \ i_2 \leftarrow \text{MaxR}[w]
6: \text{if} \ i_1 \text{ does not intersect } i_2 \text{ then}
7: \text{Weight}[r(i_2)] \leftarrow \min\{\text{Weight}[r(i_2)], \text{Weight}[d] + 3\}
8: \text{else}
9: \text{Weight}[2n] \leftarrow \min\{\text{Weight}[2n], \text{Weight}[d] + 1\}

Add-intervals-third-choice(d)
1: \ w \leftarrow \text{MinR}[d][1]
2: \text{if} \ w \neq \text{NIL} \text{ then}
3: \ i_2 \leftarrow \text{MaxR}[w]
4: \text{Weight}[r(i_2)] \leftarrow \min\{\text{Weight}[r(i_2)], \text{Weight}[d] + 2\}
5: \text{else}
6: \text{Weight}[2n] \leftarrow \min\{\text{Weight}[2n], \text{Weight}[d]\}

Roman-domination-on-interval-graphs
1: construct the data structures \ D, \text{MinR} \text{ and } \text{MaxR} by calling the procedures \ SIRE, \ ILRE and IGRE, respectively.
2: \text{for} \ i = 1 \text{ to } 2n \text{ do}
3: \text{Weight}[i] \leftarrow 2n
4: \text{Weight}[0] \leftarrow 0
5: \text{Add-intervals-first-choice}(0)
6: \text{Add-intervals-second-choice}(0)
7: \text{Add-intervals-third-choice}(0)
8: \text{for} \ i = 1 \text{ to } 2n \text{ do}
9: \text{if} \ D[i] \neq \text{NIL and Weight}[r(D[i])] \neq 2n \text{ then}
10: \text{Add-intervals-first-choice}(r(D[i]))
11: \text{Add-intervals-second-choice}(r(D[i]))
12: \text{Add-intervals-third-choice}(r(D[i]))
13: \text{return } \gamma_R(G) = \text{Weight}[2n]

Theorem 4. The Roman domination problem can be solved in \mathcal{O}(n) time on interval graphs.

Proof. The correctness of Roman-domination-on-interval-graphs follows from the lemmas stated in Section 2.2.

We note that it takes linear time to construct \ D, \text{MinR} \text{ and } \text{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-domination-on-interval-graphs is dominated by the for loop from line 8 to line 12. Therefore, the complexity of the algorithm is \mathcal{O}(n). □
4 Roman domination in 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 \( \gamma_R(G) \) 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 on page 3 the Roman domination function 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 function of \( G \) in \( O(n(m + n)) \) time, where \( n \) and \( m \) are the numbers of the vertex and edge sets 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 \( T \) [12].

In \( T \), each leaf represents a vertex of \( G \) and each internal node represents either join or 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 \( T \) is a join node. Since \( G \) is connected, the root of \( T \) is a join node. We may assume that \( T \) is a binary tree. For a node \( v \), let \( T_v \) denote the subtree of \( T \) rooted at \( v \). Let \( G_v \) denote the subgraph defined by \( T_v \). Now, our algorithm is as follows.

For a cograph \( G \), we traverse its corresponding cotree \( T \) from leaves to 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., \( \gamma_R(G_w) = 1 \). Now let us consider an internal node \( u \) in \( T \), 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 \( \gamma_R(G_l) \leq \gamma_R(G_r) \).

1. \( \gamma_R(G_l) = \gamma_R(G_r) \). 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. \( \gamma_R(G_l) < \gamma_R(G_r) \). 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 the 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 \( \gamma_R(G) \leq 4 \) 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.
The definition of a graph $\gamma_R(G_w) = 1$. Thus, $(\{w\}, \emptyset)$ is the Roman pair of $G_w$. Assume that for any node $v$ in $T$ 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 child in $T$, 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 $\gamma_R(G_l) \leq \gamma_R(G_r)$. By the definition, every vertex in $G_r$ is adjacent to any vertex of $G_l$. If $V_2(G_l)$ is 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 $\gamma_R(G_l) \leq \gamma_R(G_r)$, 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 $v_l$ dominate $G_r$ and $v_r$ dominate $G_l$, $(\emptyset, \{v_l, v_r\})$ is a Roman pair of $G_u$. These 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. \hfill \qed

Remark 4. In [3] a graph $G$ is called Roman if $\gamma_R(G) = 2\gamma(G)$. 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 [3] it is posed as an open problem to determine Roman graphs other than trees\footnote{A constructive characterization of Roman trees is given in [14].}, 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\footnote{Any induced $P_4$ would lead to an induced $P_4$ in $G$}, and furthermore for every vertex $x$ in $H$, $\gamma(H) = \gamma(H - x)$.

5 Roman domination in 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 nearly identical to those in [19] by Kratsch, and in [10] by Fomin, Kratsch and Müller.

We begin by providing some results about AT-free graphs and then we give the definition of a $d$-octopus.
Definition 5. We say that three vertices $x$, $y$ and $z$ of a graph $G = (V, E)$ form an asteroidal triple, AT as a shorthand, if

1. $\{x, y, z\}$ is an independent set,
2. for any two of the three vertices there is a path between them that avoids the neighbourhood of the third.

A graph is said to be AT-free if it does not contain an AT.

Definition 6. $(x, y)$ is a dominating pair of the graph $G = (V, E)$, if

1. $x, y \in V$,
2. the vertex set of any path between $x$ and $y$ in $G$ is a dominating set in $G$.

Theorem 6 (see [7]). 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 in 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 (see [19]). Every connected AT-free graph has a DSP.

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

1. $W$ 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 $G$ and $T$ is the union of the paths $P_1, P_2, \ldots, P_d$.

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

Remark 5. We note that the problem “Given a graph $G$ and an integer $d$, decide if $G$ has a $d$-octopus” is NP-complete (see [10]).

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

The following result is a Roman domination version of Lemma 33 in [10].

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

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

Although AT-free graphs are 1-octopus graphs, we have the following stronger result about Roman domination on AT-free graph. Our result is similar to Kratsch’s Theorem 4 in [19].
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 \( H_0, H_1, \ldots, H_l \) are the levels of the BFS-tree with the root \( x \), then \( G \) has a Roman pair \((V_1, V_2)\) such that:

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

5.1 The polynomial time algorithm

Our algorithm is similar to the one described in [19]. It 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 \( S \subseteq \bigcup_{j=0}^{l-1} H_j \) 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 5 for connected AT-free graphs and 7\(d-1\) for graphs with a \(d\)-octopus.

Now we present a variant of the algorithm in [19]. The algorithm \( rp_w(G) \), where \( w \) 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 \( w \) vertices of \( V_2 \) belong to any three consecutive levels of the BFS-tree which has \( x \) as a root, then \( rp_w(G) \) outputs a Roman pair for \( G \).

1: \( D \leftarrow V \)
2: \( \text{val}(D) \leftarrow |V| \) \{initialization: every vertex of \( V \) is in \( V_1 \), this is a trivial Roman dominating set\}
3: for all \( x \in V \) do
   4: \( \text{Compute the BFS-level of vertex } x \)
   5: \( H_0 = \{x\}, H_1 = N(x), \ldots, H_l = \{u \in V : d_G(x, u) = l\} \)
   6: \( i \leftarrow 1 \)
   7: Initialize the queue \( A_1 \) to contain an ordered triple \((S, S, \text{val}(S))\) for all nonempty subsets \( S \) of \( N[x] \) satisfying \( |S| \leq w \) with \( \text{val}(S) \leftarrow 2|S| \)
   8: Add to the queue \( A_1 \) the ordered triple \((\emptyset, \emptyset, 1)\)
9: while \( A_i \neq \emptyset \) and \( i < l \) do
   10: \( i \leftarrow i + 1 \)
   11: for all triples \((S, S', \text{val}(S'))\) in the queue \( A_{i-1} \) do
   12: \( \text{for all } U \subseteq H_i \) with \(|S \cup U| \leq w \) do
   13: \( R \leftarrow (S \cup U) \setminus H_{i-2} \)
   14: \( R' \leftarrow S' \cup U \)
   15: \( \text{val}(R') \leftarrow \text{val}(S') + 2|U| + |H_{i-1} \setminus N(S \cup U)| \)
   16: if there is no triple in \( A_i \) with first entry \( R \) then
   17: \( \text{Insert } (R, R', \text{val}(R')) \) in the queue \( A_i \)
   18: if there is a triple \((P, P', \text{val}(P'))\) in \( A_i \) such that \( P = R \) and \( \text{val}(R') < \text{val}(P') \) then
Replace \((P, P', \text{val}(P'))\) in \(A_i\) by \((R, R', \text{val}(R'))\).

20: Among all triples \((S, S', \text{val}(S'))\) in the queue \(A_i\), determine one with minimum value \(v = \text{val}(S') + |H_i \backslash N[S]|\), say \((B, B', \text{val}(B'))\)

21: if \(v < \text{val}(D)\) then
22: \(D \leftarrow B'\)
23: \(\text{val}(D) \leftarrow v\)
24: Output \((V_1, V_2) = (V \backslash N[D], D)\)

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

Proof. The analysis of the running time is the same as the Theorem 5 in [19]. In the following, we prove the correctness of the algorithm. For each triple \((S, S', \text{val}(S'))\), the set \(S'\) represents a subsolution corresponding to \(S\) with weight \(\text{val}(S')\). However, notice that the set \(S'\) does not affect the running time of the algorithm and the main purpose of storing subsolutions \(S'\) is to make it easier in the construction of a Roman pair. For an implementation of the algorithm, using a suitable pointer structure could be efficient.

We claim that for any triple \((S, S', \text{val}(S'))\) in the queue \(A_i\), \(i \in \{1, 2, \ldots, l\}\), we have \(S = S' \cap (H_{i-1} \cup H_i), \text{val}(S') = 2|S'| + |T|\) and \((\bigcup_{j=0}^{i-1} H_j) \subseteq (N[S'] \cup T)\) with \(T = (\bigcup_{j=0}^{i-1} H_j) \backslash N[S']\). This is true for \(i = 1\). By the initialization of \(A_i\), for all triples \((S, S', \text{val}(S))\) \(\in A_1\) we have \(S = S', \emptyset \subseteq S \subseteq N[x]\). Thus \(\{x\} = H_0 \subseteq N[S]\). Suppose the claim is true for \(i - 1 \in \{1, 2, \ldots, l - 1\}\). By the construction of algorithm, the triple \((R, R', \text{val}(R'))\) is in \(A_i\) only if there is a triple \((S, S', \text{val}(S'))\) in \(A_{i-1}\) and a subset \(U\) with \(|S \cup U| \leq w\) such that \(R = (S \cup U) \backslash H_i-2, R' = S' \cup U\) and \(\text{val}(R') = \text{val}(S') + 2|U| + |H_{i-1} \backslash N[S \cup U]|\).

Consequently, \(R = R' \cap (H_{i-1} \cup H_i), \text{val}(R') = 2|R'| + |T|\) and \((\bigcup_{j=0}^{i-1} H_j) \subseteq (N[R'] \cup T)\) with \(T = (\bigcup_{j=0}^{i-1} H_j) \backslash N[R']\).

Therefore, for any triple \((S, S', \text{val}(S))\) \(\in A_i\), \((V_1, V_2) = (V \backslash N[S'], S')\) is a Roman dominating set of \(G = (V, E)\). Consequently, for any Roman pair \((V_1, V_2)\) of \(G\) such that at most \(w\) vertices of \(V_2\) belong to any three consecutive BFS-levels of \(x\), there will be a triple \((S, S', \text{val}(S'))\) in \(A_i\) corresponding to \((V_1, V_2)\), when the algorithm checks all BFS-levels of \(x\). Hence the output of the algorithm is a Roman pair. \(\square\)

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. Combine Theorems 7 and 9, \(rp_{7d+1}(G)\) computes a Roman pair for a graph that is known to have a \(d\)-octopus. This algorithm take time in \(O(n^{7d+1})\).

From the algorithm \(rp_w(G)\) we see that if the root of a \(d\)-octopus is known, we shall gain a factor of \(n\) in the running time.
A graph with a DSP is a graph of 1-octopus. Therefore, there is an \(O(n^8)\)
algorithm to calculate a Roman pair for a graph with a DSP. But a DSP can be calculated in time \(O(n^3m)\) if a graph has a DSP (see [8, 19]), so we can modify the algorithm \(\text{rp}_w(G)\) by preprocessing the root to obtain an \(O(n^7)\)-time algorithm.

Analogously, \(\text{rp}_5(G)\) computes a Roman pair for a given AT-free graph. There is an \(O(n^3)\)-time recognition algorithm and a linear time algorithm to calculate a dominating pair. Modify the algorithm \(\text{rp}_5(G)\) by preprocessing the dominating pair will give us an \(O(n^8)\) algorithm (see [6, 19]). \(\square\)

6 Conclusion

We have provided, in this paper, a linear time algorithm for computing a Roman pair of an interval graph which solves an open problem raised in [3]. We have also provided a linear time algorithm to compute a Roman pair of a cograph. We extend the polynomial time algorithms developed in [19] to compute Roman pairs of AT-free graphs and graphs with a 1-octopus.

References

Appendix: Proofs of Theorems 7 and 8

The proof of Theorem 7:
This proof is almost the same as the proof of Lemma 33 in [10]. We start with a Roman pair \((V_{10}, V_{20})\) of a graph \(G\). If it satisfies the required property, then we are done. Otherwise, we construct a sequence of Roman pairs \((V_{11}, V_{21}), (V_{12}, V_{22}), \ldots\), until we find one which satisfies the required property.

Suppose \((V_r, V_{2r})\) is a Roman pair of \(G\) such that \(V_{2r}\) does not satisfy the required property. Let \(Q_r = \{(i, j) : |V_{2r} \cap \bigcup_{s=i}^{r+j} H_s| \geq (j + 5)d\}\). Then \(Q_r \neq \emptyset\). We choose \((i'_r, j'_r) \in Q_r\) such that \(i'_r = \min\{i : (i, j) \in Q_r\}\) and according to \(i'_r, j'_r = \max\{j : (i'_r, j) \in Q_r\}\).

Since \(H_0, H_1, \ldots, H_{l}\) are the levels of a BFS-tree of \(G\), this ensures that any neighbour of a vertex in \(V_{2r} \cap \bigcup_{s=i'_r}^{r+j'_r} H_s\) belongs to one of the levels \(H_{i'_r - 1}, H_{i'_r}, \ldots, H_{i'_r + j'_r + 1}\). Let \(V_{1r+1} = V_{1r}\) and \(V_{2r+1} = (V_{2r} \backslash \bigcup_{s=i'_r}^{r+j'_r} H_s) \cup (D \cap \bigcup_{s=i'_r}^{r+j'_r+2} H_s)\). Then \(V_{2r+1}\) dominates \(V \setminus V_{1r+1}\). Moreover, \(|V_{2r} \cap \bigcup_{s=i'_r}^{r+j'_r} H_s| \geq (j_r + 5)d\) and \(|D \cap \bigcup_{s=i'_r}^{r+j'_r+2} H_s| \leq (j_r + 5)d\), thus \(|V_{2r}|| \geq |V_{2r+1}||. Therefore, \((V_{1r+1}, V_{2r+1})\) is also a Roman pair of \(G\).

We may assume that \(i'_r > 2\) and \(i'_r + j'_r < l - 2\) since \(H_s = \emptyset\) for \(s < 0\) or \(s > l\).

The replacement of \((V_{1r}, V_{2r})\) by \((V_{1r+1}, V_{2r+1})\) is called an exchange step. It remains to show that the sequence of Roman pair \((V_{10}, V_{20}), (V_{11}, V_{21}), \ldots\) which do not satisfy the required property is finite. To do this we prove \(i_r + j_r + 2 < i_{r+1}\) for all steps of the construction with \(Q_{r+1} \neq \emptyset\).

Suppose \(i_{r+1} \leq i_r + j_r + 2\). The choice of \(i_r\) and \(j_r\) implies that \(i_{r+1} + j_{r+1} \geq i_r\) since \(V_{2r} \cap (\bigcup_{s=i_r}^{r+j_r} H_s) = V_{2r+1} \cap (\bigcup_{s=i_r}^{r+j_r} H_s)\). Next we have \(i_{r+1} + j_{r+1} \geq i_r + j_r + 2\) since \(V_{2r+1} \cap (\bigcup_{s=i_r}^{r+j_r+2} H_s) \geq d\) for all \(s\) with \(i_r - 2 \leq s \leq i_r + j_r + 2\). This implies \(|V_{2r+1} \cap (\bigcup_{s=i_r}^{r+j_r+1} H_s)| = |V_{2r} \cap (\bigcup_{s=i_r}^{r+j_r+1} H_s)|\), which contradicts the choice of \(i_r\) and \(j_r\). Therefore, \(i_{r+1} > i_r + j_r + 2\).

The proof of Theorem 8:
Again, the proof is nearly the same as the proof of Theorem 4 in [19]. Let \(G\) be a connected AT-free graph.

There is a linear time algorithm [18] to compute for any given connected AT-free graph \(G\) a path \(P = (x = x_0, x_1, \ldots, x_d = y)\) such that:

1. \(x_i \in H_i, \forall i \in \{0, 1, \ldots, l\}\),
2. the set of vertices \(V(P)\) of the path is a dominating set for \(G\),
3. each vertex \(z \in H_i, i \in \{0, 1, \ldots, l\}\), is adjacent to either \(x_i\) or \(x_{i-1}\).

We start with a Roman pair of \((V_{10}, V_{20})\) of \(G\). If \(V_{20}\) does not satisfy the required property, then we will construct a sequence of Roman pairs \((V_{11}, V_{21}), (V_{12}, V_{22}), \ldots\), until we find one which satisfies the required property.

Suppose \((V_{1r}, V_{2r})\) is a Roman pair of \(G\) such that \(V_{2r}\) does not satisfy the required property. Let \(Q_r = \{(i, j) : |V_{2r} \cap \bigcup_{s=i}^{r+j} H_s| \geq j + 4\}\). Then \(Q_r \neq \emptyset\).
We select \((i'_r, j'_r) \in Q_r\) such that \(i'_r = \min \{i : (i, j) \in Q_r\}\) and according to \(i'_r, j'_r = \max \{j : (i'_r, j) \in Q_r\}\).

Every neighbour of a vertex in \(V_{2r} \cap \bigcup_{s=i'_r}^{j'_r} H_s\) belongs to one of the levels \(H_{i'_r-1}, H_{i'_r}, \ldots, H_{i'_r+j'_r+1}\). Let \(A = \{x_{i'_r-2}, x_{i'_r-1}, \ldots, x_{i'_r+j'_r+1}\}\). Since every vertex \(z \in H_k\), for \(k \in \{i'_r - 1, i'_r, \ldots, i'_r + j'_r + 1\}\), is adjacent to \(x_{k-1}\) or \(x_k\), we must have \(\bigcup_{s=i'_r}^{j'_r} H_s \subseteq N[A]\). Let \(V_{1r+1} = V_{1r}\) and \(V_{2r+1} = (V_{2r} \setminus \bigcup_{s=i'_r}^{j'_r} H_s) \cup A\). Then \(|V_{2r} \cap \bigcup_{s=i'_r}^{j'_r} H_s| \geq j'_r + 4\) and \(|A| = j'_r + 4\), thus \(|V_{2r}| \geq |V_{2r+1}|\). Consequently, \((V_{1r+1}, V_{2r+1})\) is a Roman pair of \(G\).

We may assume that \(i'_r > 2\) and \(i'_r + j'_r < l - 2\) since then \(A = \{x_0, x_1, \ldots, x_{i'_r+j'_r+2}\}\) or \(\{x_{i'_r-2}, x_{i'_r-1}, \ldots, x_{d}\}\) contains less than \(j'_r + 5\) vertices. Therefore the Roman set \((V_{1r+1}, V_{2r+1})\) would have smaller weight than the Roman set \((V_{1r}, V_{2r})\), which is a contradiction.

If \((V_{1r+1}, V_{2r+1})\) has the required property, then \(G\) has a Roman pair with the required property. Otherwise, we consider \(Q_{r+1} = \{(i, j) : |V_{2r+1} \cap \bigcup_{s=i}^{j+1} H_s| \geq j + 4\} \neq \emptyset\). Suppose \((i, j) \in Q_{r+1}\) with \(i \leq i_r\). Then \(i + j \geq i'_r - 2\), otherwise we have \((i, j) \in Q_r\), contradicting the previous choice of \(i'_r\). By construction \(|V_{2r+1} \cap H_s| \geq 1\) for all \(s \in \{i'_r - 2, i'_r - 1, \ldots, i'_r + j'_r + 1\}\). Thus \((i, j) \in Q_{r+1}\) with \(i \leq i_r\) and \(i + j \geq i'_r - 2\) implies that there is a \(j'\) such that \((i, j') \in Q_{r+1}\) and \(i + j' \geq i'_r + j'_r + 1\). By the construction of \(V_{2r+1}\), we have \(|V_{2r+1} \cap \bigcup_{s=i'_r}^{j'_r+1} H_s| = |V_{2r} \cap \bigcup_{s=i'_r}^{j'_r+1} H_s|\) and thus \((i, j') \in Q_r\), contradicting the previous choice of either \(i'_r\) or \(j'_r\). Consequently \(i'_r + 1 = \min \{i : (i, j) \in Q_{r+1}\} > i'_r\).

Hence starting with a Roman pair \((V_{10}, V_{20})\) of \(G\) we obtain a Roman pair \((V_1, V_2)\) having the required property after at most \(d\) exchange steps.

\(\Box\)