Efficient algorithms for Roman domination on some classes of graphs
Mathieu Liedloff\textsuperscript{a}, Ton Kloks\textsuperscript{b,1}, Jiping Liu\textsuperscript{b,2}, Sheng-Lung Peng\textsuperscript{c,}\textsuperscript{*}

\textsuperscript{a} Universit\'e Paul Verlaine – Metz, LITA, 57045 Metz Cedex 01, France
\textsuperscript{b} Department of Mathematics and Computer Science, The University of Lethbridge, Alberta, T1K 3M4, Canada
\textsuperscript{c} Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien 974, Taiwan

Received 28 December 2005; received in revised form 7 January 2008; accepted 11 January 2008
Available online 7 March 2008

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 first answer an open question mentioned in [E.J. Cockayne, P.A. Dreyer Jr., S.M. Hedetniemi, S.T. Hedetniemi, Roman domination in graphs, Discrete Math. 278 (2004) 11–22] by showing that the Roman domination number of an interval graph can be computed in linear time. We then show that the Roman domination number of a cograph (and a graph with bounded cliquewidth) can be computed in linear time. As a by-product, we give a characterization of Roman cographs. It leads to a linear-time algorithm for recognizing Roman cographs. Finally, we show that there are polynomial-time algorithms for computing the Roman domination numbers of $\mathcal{AT}$-free graphs and graphs with a $d$-octopus.

\textsuperscript{©} 2008 Elsevier B.V. All rights reserved.

Keywords: Graph algorithms; Roman domination; Interval graphs; Cographs; AT-free graphs

1. Introduction
Let $G = (V, E)$ be a simple and undirected 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 Roman domination problem for $G$ is to find a Roman dominating function of $G$ with minimum weight. The minimum weight of a Roman dominating function on $G$ is called the Roman domination number of $G$ and is denoted by $\gamma_R(G)$.\textsuperscript{\textcopyright}
Roman domination was introduced in [2] as a new variety of the classical domination problem having both historical and mathematical interest, particularly in the field of server placement [19]. We refer to [2–4,10,13–15,21,22] 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 [2] that the Roman domination problem on trees can be solved in linear time and it remains NP-complete when restricted to split graphs, bipartite graphs, and planar graphs. Linear-time algorithms for the problem on block graphs and bounded treewidth graphs are proposed in [16,20]. The complexity of the Roman domination problem when restricted to interval graphs was mentioned as an open question in [2].

In this paper we show that there are linear-time algorithms to compute the Roman domination number for interval graphs and cographs. Moreover, we prove that the Roman domination problem can be expressed as a LinEMSOL(τ1) optimization problem. The immediate consequence is that the Roman domination problem can be solved in linear time on graphs \( G \) with a bounded cliquewidth \( k \), provided that a \( k \)-expression of \( G \) is also a part of the input. We also show that there are polynomial-time algorithms for computing the Roman domination numbers of \( \text{AT} \)-free graphs and graphs with a \( d \)-octopus.

Let \( \gamma(G) \) denote the domination number of \( G \) (will be defined later). A graph \( G \) is called Roman if \( \gamma_R(G) = 2\gamma(G) \). A constructive characterization of Roman trees is given in [13]. Our result on cographs also provides a characterization for Roman cographs. It implies that Roman cographs can be recognized in linear time. This result answers another open question given in [2], i.e., determining Roman graphs other than trees.

The paper is organized as follows. Section 2 gives some preliminaries about our problem. A dynamic programming algorithm for the problem on interval graphs is presented in Section 3. The results for cographs and graphs with bounded cliquewidth are presented in Section 4. In Section 5, we show that there are polynomial-time algorithms for computing the Roman domination numbers of \( \text{AT} \)-free graphs and graphs with a \( d \)-octopus. A conclusion is provided in the final section.

2. Preliminaries

Let \( G = (V,E) \) be a simple and undirected graph. Let \( \overline{G} \) be the complement of \( G \), i.e., \( \overline{G} = (V, \{xy : x, y \in V \text{ and } xy \notin E\}) \). For a subset \( W \) of \( V \), let \( G[W] \) be the subgraph induced by \( W \), i.e., \( G[W] = (W, \{xy : x, y \in W \text{ and } xy \in E\}) \). 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 \( x \). For a subset \( W \subseteq V \) we write \( N(W) = \bigcup_{x \in W} N(x) \setminus W \) and \( N[W] = \bigcup_{x \in W} N[x] \). The distance \( d_G(x, y) \) between two vertices \( x \) and \( y \) is the length of the shortest path joining these two vertices.

For two sets \( A \) and \( B \) we write \( A + B \) and \( A - B \) instead of \( A \cup B \) and \( A \setminus B \) respectively. For an element \( x \) we write \( A - x \) instead of \( A \setminus \{x\} \) and \( A + x \) instead of \( A \cup \{x\} \). For a vertex \( x \) we write \( G - x \) rather than \( G[V \setminus \{x\}] \).

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 called 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 ([2]).** \( \gamma(G) \leq \gamma_R(G) \leq 2\gamma(G) \).

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

**Definition 3.** 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 \( G \).

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

\(^3\) A fortiori, \( G \) is a graph with at least two vertices.
(1) \( V_1 \) is a 2-packing and 
(2) there is no edge between \( V_1 \) and \( V_2 \).

**Theorem 5** ([2]). For any non-trivial connected graph \( G \),
\[
\gamma_R(G) = \min\{|S| + 2\gamma(G - S) \mid S \text{ is a 2-packing of } G\}.
\]

**Remark 6.** 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 7.** We call \((V_1, V_2)\) a Roman pair of a graph \( G \) if \((V_1 = \{x : f(x) = 1\}, V_2 = \{x : f(x) = 2\})\) is a solution induced by a minimum weight Roman dominating function \( f \) of \( G \).

**Remark 8.** If we know the set \( V_2 \) which is induced by a minimum weighted Roman dominating function of a graph \( G = (V, E) \), we can deduce that \( V_1 = V - N[V_2] \).

We refer the reader to [1] 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 a connected graph. Clearly, if \( G \) is disconnected then \( \gamma_R(G) \) is the sum of the Roman domination numbers of its components.

**Definition 9.** A graph \( G = (V, E) \) is an interval graph if there exists a set \( F = \{I_v \mid v \in V\} \) of intervals of the real line such that \( I_u \cap I_v \neq \emptyset \) if and only if \( uv \in E \). The set \( F \) is also called the interval model of \( G \).

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 \( I_v \), respectively. An interval model of \( G \) is normalized if \( \bigcup_{v \in V} \{l(v), r(v)\} = \{1, 2, \ldots, 2n\} \). In the following we assume that a normalized interval model of \( G \) is part of the input.

3.1. Structure of an optimum solution

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

**Definition 10.** For two intervals \( I_v \) and \( I_u \) we say that \( I_v \) is properly contained in \( I_u \) if \( l(u) < l(v) < r(v) < r(u) \).

**Lemma 11.** For every interval graph there exists a Roman pair \((V_1, V_2)\) such that no interval of \( V_2 \) is properly contained in another interval of \( V_2 \).

**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 \((V_1, V_2 - i)\) is a Roman pair which contradicts the fact that \((V_1, V_2)\) is a Roman pair. \( \square \)

**Lemma 12.** If \((V_1, V_2)\) is a Roman pair of an interval graph \( G \), then \( V_2 \) contains no clique of size three or more.

**Proof.** Let \( \{i_1, i_2, i_3\} \subseteq V_2 \) be a clique of size three. By Lemma 11, 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 \( N[i_2] \subseteq N[i_1] + N[i_3] \). That is, \((V_1, V_2 - i_2)\) is a Roman pair of \( G \) which is a contradiction. \( \square \)

**Lemma 13.** If \((V_1, V_2)\) is a Roman pair of an interval graph \( G \), then the connected components induced by \( V_2 \) are paths.

**Proof.** By Lemma 11, 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 12, the lemma holds. \( \square \)
We can use the 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 14.** Let $(V_1, V_2)$ be a Roman pair of an interval graph $G$. Let $J$ be a subset of $V_1$. We denote by $I(J)$ (respectively, by $r(J)$) the leftmost (respectively, the rightmost) end point of $J$. The intervals of $J$ are consecutive, or shortly $J$ is consecutive, if and only if there is no end point of an interval of $V_2$ between $l(J)$ and $r(J)$.

**Lemma 15.** For any Roman pair $(V_1, V_2)$ of an interval graph $G$ with $V_1$ being a 2-packing, there is no consecutive subset $J \subseteq V_1$ containing more than two intervals.

**Proof.** Let $(V_1, V_2)$ be a Roman pair with $V_1$ being a 2-packing. Assume that $\{a, b, c\}$ is a consecutive subset of $V_1$ and, without loss of generality, let $l(a) < r(a) < l(b) < r(b) < l(c) < r(c)$. Since $G$ is connected it follows that $N(b)$ is not empty. Moreover, since $(V_1, V_2)$ is of minimum weight and $V_1$ is a 2-packing, for all $v \in N(b)$, $v$ is neither in $V_2$ nor in $V_1$. Then, given a vertex $v$ of $N(b)$, there must exist a $w \in N(v)$ such that $w \in V_2$. Since $V_1$ is a 2-packing, $v$ does not intersect $a$ and $c$. Thus one end point of $w$ should be between $r(a)$ and $l(c)$, contradicting $\{a, b, c\}$ to be consecutive. □

By Lemmas 13 and 15, we only consider Roman pairs $(V_1, V_2)$ of $G$ with $V_1$ being a 2-packing, $V_1$ containing no consecutive subset of size more than two, and $V_2$ inducing a set of paths. Note that if one of these Roman pairs $(V_1, V_2)$ fulfills $V_2 = \emptyset$, then it implies that $G$ has at most one vertex. It is quite simple to compute a Roman pair of such a graph. In the next, we suppose that $V_2 \neq \emptyset$.

Given a Roman pair $(V_1, V_2)$ and an integer $d$ corresponding to a right end point of an interval of $V_2$, the pair $(V_1^d, V_2^d)$ is defined by $(V_1 \cap Z, V_2 \cap Z)$ where $Z = \{v \in V : r(v) \leq d\}$. Such a pair is called a sub-solution and is an optimal solution for the graph $G[S]$ where $S = \{v \in V : l(v) \leq d\}$. For convenience, we let $(V_1^0, V_2^0) = (\emptyset, \emptyset)$.

The next theorems derive a recurrence relation for computing an optimal solution (i.e., a Roman pair).

**Theorem 16.** Let $(V_1, V_2)$ be a Roman pair of an interval graph $G$ with $V_1$ being a 2-packing. Let $v$ be an interval of $V_2$ and let $d' = r(v)$. Let $d$ be the largest right end point in $V_2$ smaller that $d'$. The value of $d$ is set to 0 if $d'$ is the smallest right end point in $V_2$. The sub-solution $(V_1^d, V_2^d)$ is one of the following:

1. $(V_1^d + \{j_1, j_2\}, V_2^d + v)$ where $d < l(j_1) < r(j_1) < l(j_2) < r(j_2) < l(v)$.
2. $(V_1^d + j_1, V_2^d + v)$ where $d < l(j_1) < r(j_1) < l(v)$.
3. $(V_1^d, V_2^d + v)$.

**Proof.** By Lemmas 13 and 15, $(V_1^d, V_2^d)$ is one of the three given cases. □

Finally, by Lemma 15, we have the following theorem.

**Theorem 17.** Let $(V_1, V_2)$ be a Roman pair of an interval graph $G$ with $V_1$ being a 2-packing. Let $d$ be the largest right end point in $V_2$. Then $(V_1, V_2)$ is one of the following:

1. $(V_1^d + \{j_1, j_2\}, V_2^d)$ where $d < l(j_1) < r(j_1) < l(j_2)$.
2. $(V_1^d + j_1, V_2^d)$ where $d < l(j_1)$.
3. $(V_1^d, V_2^d)$.

Theorems 16 and 17 provide us a way to find an optimal solution by dynamic programming technique. Using a traditional implementation, for a right end point $d'$, we should find a suitable $d < d'$ according to Theorem 16. That is, we can obtain a sub-solution $(V_1^{d'}, V_2^{d'})$ by a suitable sub-solution $(V_1^d, V_2^d)$. However, by doing so, we must add a dummy interval in $V_2$ such that it can take care of the case of Theorem 17. To avoid this situation, we implement it in another way. In other words, for an integer $d$ and a corresponding sub-solution $(V_1^d, V_2^d)$, we extend it to $d' (>d)$ such that $(V_1^{d'}, V_2^{d'})$ is a sub-solution that satisfies Theorems 16 and 17. We say that $(V_1^{d'}, V_2^{d'})$ is an extension of $(V_1^d, V_2^d)$. 

"
Let an array of intervals arise since there is no extension of intervals such that $r(i) = \min \{r(i) : l(i) > d \land i \neq i_1 \land i \neq i'_1\}$. If the three intervals $i_1, i'_1, w$ exist and $w$ intersects $i_1$, then there is no extension $(V_1^{d'}, V_2^{d'})$ of $(V_1^d, V_2^d)$, where $d' > d$, such that $V_1^{d'}$ is a 2-packing containing $\{i_1, i'_1\}$ as two consecutive intervals.

**Proof.** By the definition, we have $l(w) < r(i_1) < r(w)$. Suppose that there exists an extension $(V_1^{d'}, V_2^{d'})$ of $(V_1^d, V_2^d)$, $d' > d$, such that $V_1^{d'}$ is a 2-packing containing $i_1$ and $i'_1$. It follows that $w \not\in N(i_1')$, i.e., $l(w) < r(i_1) < r(w) < l(i'_1)$. Since $w$ needs a neighbor in $V_2$, there should be a vertex $v \in V_2^{d'}$ being adjacent to $w$. A contradiction arises since $i_1$ and $i'_1$ are two consecutive intervals.

### 3.2. Preprocessing data

In order to obtain a linear-time algorithm, we first preprocess the input data. The objective is to guarantee that each of the following operations can be done in constant time.

- find $i, j, k$ such that $r(i) = \min \{r(v) : l(v) > d\}$, $r(j) = \min \{r(v) : l(v) > d \land v \neq i\}$ and $r(k) = \min \{r(v) : l(v) > d \land v \neq i \land v \neq j\}$ for a fixed $d$.
- find $i$ such that $r(i) = \max \{r(v) : v \in N[x]\}$ for an interval $x$.
- 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 $i \in N[y]$).

**Sort Intervals According to Their Right End Points (SIRE).**

The collection $\mathcal{I}$ of $n$ intervals is given by a normalized interval model. We sort these intervals in time $O(n)$ using a bucket sort.

**Procedure SIRE(\mathcal{I})**

**Data:** An interval collection $\mathcal{I}$ and its normalized model.

**Result:** An array $D$ containing the intervals sorted according to their right end points.

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

$[D[i] \gets \text{NIL}]]$

**for** $i = 1$ to $n$ **do**

$[D[r(i)] \gets i]]$

**Example 19.** For the collection of intervals as shown in Fig. 1, we obtain the following array $D$: 

```
  1  3  4  5  10
  2  7  9  5  12
  8  6  11
```

Fig. 1. An interval collection given by a normalized model.
Find Three Intervals with Lowest Right End Points (ILRE).

By using the array $D$, we build a $(3 \times 2n + 1)$-array $\text{MinR}$. For each integer $i \in \{0, 1, \ldots, 2n\}$, we obtain that $\text{MinR}[1][i] = v_1$, $\text{MinR}[2][i] = v_2$, and $\text{MinR}[3][i] = v_3$ such that (1) $i < \min(l(v_1), l(v_2), l(v_3))$; (2) $r(v_1) < r(v_2) < r(v_3)$ and there is no interval $v \in I - \{v_1, v_2, v_3\}$ with $i < r(v) < r(v_3)$. Note that if there is no such interval in $\text{MinR}[j][i]$, then $\text{MinR}[j][i]$ is set to $\text{NIL}$.

**Procedure ILRE($I$, $D$)**

**Data:** An interval collection $I$ and its normalized model. The array $D$ obtained from procedure SIRE.

**Result:** An array $\text{MinR}$ fulfilling the previously explained properties.

```plaintext
for i = 0 to 2n do
    for j = 1 to 3 do
        $\text{MinR}[j][i] \leftarrow \text{NIL}$
    indexMinR $\leftarrow 0$
    for i = 1 to 2n do
        if $D[i] \neq \text{NIL}$ then
            while $l(D[i]) > indexMinR$ do
                $\text{MinR}[1][indexMinR] \leftarrow D[i]$
                indexMinR $\leftarrow indexMinR + 1$
        for j = 2 to 3 do
            prev $\leftarrow 0$
            indexMinR $\leftarrow 0$
            for i = $r(\text{MinR}[j - 1][0]) + 1$ to 2n do
                if $D[i] \neq \text{NIL}$ then
                    if prev $\neq \text{MinR}[j - 1][indexMinR]$ and prev $\neq 0$ then
                        prev $\leftarrow \text{MinR}[j - 1][indexMinR]$
                    while $l(D[i]) > indexMinR$ and ($\text{MinR}[j - 1][indexMinR] = prev$ or prev $= 0$) do
                        $\text{MinR}[j][indexMinR] \leftarrow D[i]$
                        prev $\leftarrow \text{MinR}[j - 1][indexMinR]$
                    indexMinR $\leftarrow indexMinR + 1$
```

**Example 20.** For the previous collection, we obtain the following array $\text{MinR}$:

<table>
<thead>
<tr>
<th>$D$ :</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></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>NIL</td>
</tr>
</tbody>
</table>

| $\text{MinR}$ : | 4 | 2 | 1 | 6 | 6 | 5 | 5 | 5 | NIL| NIL| NIL| NIL| NIL|
|                | 2 | 1 | 6 | 5 | 5 | NIL| NIL| NIL| NIL| NIL| NIL| NIL| NIL|

Find Intervals with Greatest Right End Points (IGRE).

Finally, we compute the array $\text{MaxR}$. For each interval $i \in \{1, \ldots, n\}$, the value of $\text{MaxR}[i]$ is the interval $v$ intersecting $i$ with the greatest right end point ($v$ could be $i$ itself).
**Procedure** IGRE(I,D)  
**Data:** An interval collection $I$ and its normalized model. The array $D$ obtained from procedure SIRE.  
**Result:** An array $MinR$ fulfilling the previously explained properties.  
\[ \text{indexMaxR} \leftarrow 1 \]  
/* first, we order the intervals decreasingly according to their right end points. */  
for $i = 2n$ to 1 do  
\[ \text{if } D[i] \neq \text{NIL then} \]  
\[ D'[\text{indexMaxR}] \leftarrow D[i] \]  
\[ \text{indexMaxR} \leftarrow \text{indexMaxR} + 1 \]  
/* then, we construct the array which contains $u$ such that $r(u) = \max\{r(v) : v \in \text{N}[i]\}$ for all $i$. */  
\[ \text{indexMaxR} \leftarrow 1 \]  
for $i = 2n$ to 1 do  
\[ \text{if } D[i] \neq \text{NIL then} \]  
\[ \text{while indexMaxR} \leq n \text{ and } l(D'[\text{indexMaxR}]) \leq r(D[i]) \text{ and } l(D[i]) \leq r(D'[\text{indexMaxR}]) \text{ do} \]  
\[ \text{/* i.e., } D'[\text{indexMaxR}] \text{ is a neighbor of } D[i] */ \]  
\[ \text{MaxR}[D'[\text{indexMaxR}]] \leftarrow D[i] \]  
\[ \text{indexMaxR} \leftarrow \text{indexMaxR} + 1 \]  

**Example 21.** For the previous collection, we obtain the following array $MaxR$:

<table>
<thead>
<tr>
<th>interval:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxR:</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

### 3.3. A linear-time algorithm

Using the structure of an optimum solution described in Section 3.1, we are ready to present a linear-time algorithm for solving the Roman domination problem on interval graphs.

For an optimal solution $(V_1, V_2)$, 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 from left to right. 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 extend it to all possible extensions.

Initially, for $d = 0$, $(V_1^d, V_2^d) = (\emptyset, \emptyset)$. Then, at each step, we start with a current integer $d$ and its corresponding sub-solution $(V_1^d, V_2^d)$. We construct an extension $(V_1^{d'}, V_2^{d'})$ of $(V_1^d, V_2^d)$ corresponding to a new $d'$, where $d' > d$. According to previous theorems, there are three possible cases:

1. add two intervals $i_1$ and $i_1'$ to $V_1^d$ and one interval $i_2$ to $V_2^d$ such that $(V_1^d, V_2^d) = (V_1^d + \{i_1, i_1'\}, V_2^d + i_2)$ is a sub-solution corresponding to $d' = r(i_2)$ (see procedure Add-intervals-first-choice);
2. add one interval $i_1$ to $V_1^d$ and one interval $i_2$ to $V_2^d$ such that $(V_1^d, V_2^d) = (V_1^d + i_1, V_2^d + i_2)$ is a sub-solution corresponding to $d' = r(i_2)$ (see procedure Add-intervals-second-choice);
3. add one interval $i_2$ to $V_2^d$ such that $(V_1^d, V_2^d) = (V_1^d, V_2^d + i_2)$ is a sub-solution corresponding to $d' = r(i_2)$ (see procedure Add-intervals-third-choice).
The first case corresponds to adding two consecutive intervals to \( V_1^d \) and then starting a new path in \( V_2^d \). Note that by Lemma 18 we can avoid some extensions in this case. In the second case, we add one interval to \( V_1^d \) and begin a new path in \( V_2^d \). In the last case, we add only one interval to \( V_2^d \) which extends an existing path in \( V_2^d \) or begins a new path in \( V_2^d \). Note that it is possible that \( i_2 \) does not exist for the above three cases. In this case, the best solution will be recorded at the final position. The three extension procedures are presented as follows.

**Procedure** Add-intervals-first-choice(\( d \))

**Data:** An integer \( d \) such that a corresponding sub-solution \((V_1^d, V_2^d)\) has already been computed.

**Result:** An extension of \((V_1^d, V_2^d)\) according to the first case.

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

if \( i_1 \neq \text{NIL} \) then

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

if \( i'_1 \neq \text{NIL} \) then

if \( \text{MaxR}[i_1] \) does not intersect \( i'_1 \) then

\[
w \leftarrow \text{MinR}[2][d]
\]

if \( w = i'_1 \) then \( w \leftarrow \text{MinR}[3][d] \)

if \( w \neq \text{NIL} \) then

if \( i_1 \) does not intersect \( w \) then

\[
i_2 \leftarrow \text{MaxR}[w]
\]

if \( i_1 \) does not intersect \( i_2 \) and \( i'_1 \) does not intersect \( i_2 \) then

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

else \( \text{Weight}[2n] \leftarrow \min\{\text{Weight}[2n], \text{Weight}[d] + 2\} \)

**Procedure** Add-intervals-second-choice(\( d \))

**Data:** An integer \( d \) such that a corresponding sub-solution \((V_1^d, V_2^d)\) has already been computed.

**Result:** An extension of \((V_1^d, V_2^d)\) according to the second case.

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

if \( i_1 \neq \text{NIL} \) then

\[
w \leftarrow \text{MinR}[2][d]
\]

if \( w \neq \text{NIL} \) then

\[
i_2 \leftarrow \text{MaxR}[w]
\]

if \( i_1 \) does not intersect \( i_2 \) then

\[
\text{Weight}[r(i_2)] \leftarrow \min\{\text{Weight}[r(i_2)], \text{Weight}[d] + 3\}
\]

else \( \text{Weight}[2n] \leftarrow \min\{\text{Weight}[2n], \text{Weight}[d] + 1\} \)

**Procedure** Add-intervals-third-choice(\( d \))

**Data:** An integer \( d \) such that a corresponding sub-solution \((V_1^d, V_2^d)\) has already been computed.

**Result:** An extension of \((V_1^d, V_2^d)\) according to the third case.

\[
w \leftarrow \text{MinR}[1][d]
\]

if \( w \neq \text{NIL} \) then

\[
i_2 \leftarrow \text{MaxR}[w]
\]

\[
\text{Weight}[r(i_2)] \leftarrow \min\{\text{Weight}[r(i_2)], \text{Weight}[d] + 2\}
\]

else \( \text{Weight}[2n] \leftarrow \min\{\text{Weight}[2n], \text{Weight}[d]\} \)
By using the three extension procedures, the main algorithm is lucid. The detail is as follows.

**Algorithm** Roman-Interval($G$)

**Data:** An interval graph $G$ represented by a normalized model.

**Result:** The Roman domination number $\gamma_R(G)$.

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

**Example 22.** For the previous example, we obtain the following array $\text{Weight}$:

1. after execution of line 4:

   \[
   \begin{array}{cccccccccccc}
   & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
   \text{Weight}: & 0 & 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \\
   \end{array}
   \]

2. after execution of line 7:

   \[
   \begin{array}{cccccccccccc}
   & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
   \text{Weight}: & 0 & 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 & 3 & 12 & 12 & 12 \\
   \end{array}
   \]

3. after the iteration of $i = 7$ (lines 8–12):

   \[
   \begin{array}{cccccccccccc}
   & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
   \text{Weight}: & 0 & 12 & 12 & 12 & 12 & 12 & 12 & 2 & 12 & 12 & 3 & 12 & 4 \\
   \end{array}
   \]

4. after the iteration of $i = 10$ (lines 8–12):

   \[
   \begin{array}{cccccccccccc}
   & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
   \text{Weight}: & 0 & 12 & 12 & 12 & 12 & 12 & 2 & 12 & 12 & 3 & 12 & 3 & 3 \\
   \end{array}
   \]

The final result is recorded at the last entry, i.e., $\text{Weight}[2n]$. In this example, the weight is 3. By a backtracking, we obtain that $(V_1, V_2) = ([3], [1])$.

**Theorem 23.** The Roman domination problem can be solved in $O(n)$ time on interval graphs provided a normalized interval model as part of the input.

**Proof.** The correctness of the algorithm follows from the lemmas and theorems stated in Section 3.1.

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-Interval is dominated by the second for loop. Therefore, the complexity of the algorithm is $O(n)$. □
4. Roman domination on cographs and graphs of bounded cliquewidth

4.1. A simple linear-time algorithm for 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 connected components.

Let $G_I = (V_I, E_I)$, $G_r = (V_r, E_r)$ be two graphs such that $V_I \cap V_r = \emptyset$. We denote the union of $G_I$ and $G_r$ as the graph $G = (V_I + V_r, E_I + E_r)$. The join of $G_I$ and $G_r$ is defined as the graph $G = (V_I + V_r, E_I + E_r + \{vw : v \in V_I, w \in V_r\})$. The class of cographs can be defined by the following recursive definition:

1. The graph of a single vertex is a cograph.
2. If $G_I$ and $G_r$ are cographs, then so are the union and join of $G_I$ and $G_r$.
3. No graph is a cograph unless it can be constructed from copies of a single vertex by a finite number of applications of the operations in (2) above.

Note that if a cograph $G$ is connected then $G$ is the join of two cographs $G_I$ and $G_r$, and then any 2-packing of $G$ consists of at most one vertex since $G$ is $P_4$-free. By Theorem 5, $\gamma_R(G)$ 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 $\gamma_R(G)$ 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 $T$ [12]. In $T$, 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 $uv$ 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. Furthermore, we may assume that $T$ is a binary tree.

For a graph $G = (V, E)$ (not necessarily connected), a vertex $u$ is a universal (respectively, independent) vertex of $G$ if $u$ is adjacent to every vertex (respectively, no vertex) in $V - u$. We call $u$ almost universal if $u$ is universal in $G - v$ for a vertex $v$ with $uv \notin E$. We classify cographs $G$ according to $\gamma_R(G)$ in the following theorem.

Theorem 24. Let $G$ be a connected cograph that is a join of the two cographs $G_I = (V_I, E_I)$ and $G_r = (V_r, E_r)$. Assume that $|V_I + V_r| \geq 3$. We have

1. $\gamma_R(G) = 2$ if and only if one of $G_I$ and $G_r$ contains a universal vertex;
2. $\gamma_R(G) = 3$ if and only if one of $G_I$ and $G_r$ contains an almost universal vertex and $\gamma_R(G) \neq 2$;
3. $\gamma_R(G) = 4$ otherwise.

Proof. Since $G$ is the join of $G_I$ and $G_r$, every vertex of $V_I$ is adjacent to every vertex of $V_r$ in $G$. Thus any universal vertex $u$ in $G_I$ (or $G_r$) is also a universal vertex in $G$. Thus $(\emptyset, \{u\})$ is a Roman pair of $G$ such that $\gamma_R(G) = 2$. On the other hand, if $\gamma_R(G) = 2$, then $G$ must contain a universal vertex $u$ since $|V_I + V_r| \geq 3$. This implies that $u$ is also a universal vertex in $G_I$ (or $G_r$). It proves Statement (1).

For Statement (2) we assume that one of $G_I$ and $G_r$ contains an almost universal vertex $u$ and an independent vertex $v$. It is obvious that $(\emptyset, \{u\})$ is a Roman pair of $G$ such that $\gamma_R(G) = 3$. On the other hand, if $\gamma_R(G) = 3$, then an optimal Roman pair must be like $(\{v\}, \{u\})$ since $G$ is connected. Furthermore, $u$ and $v$ must be in one of $G_I$ and $G_r$ simultaneously. Otherwise, $(\emptyset, \{u\})$ is a Roman pair of $G$ which contradicts the assumption. Therefore, Statement (2) holds.

By letting $u \in V_I$ and $v \in V_r$, it is easy to see that $(\emptyset, \{u, v\})$ is a Roman dominating function of $G$. That is, for any connected cograph $G$, $\gamma_R(G) \leq 4$. This finishes the proof.

According to Theorem 24, we have the following algorithm for a connected cograph $G$. 
Algorithm Roman-Cograph$(G)$

Data: A connected cograph $G$ with its cotree.

Result: The Roman domination number $\gamma_R(G)$.

Let $G$ be the join of $G_l$ and $G_r$.

if $G_l$ or $G_r$ contains a universal vertex then
  return $\gamma_R(G) = 2$
else
  if $G_l$ or $G_r$ contains an almost universal vertex then
    return $\gamma_R(G) = 3$
  else
    return $\gamma_R(G) = 4$

Finally, we have the following theorem.

Theorem 25. The Roman domination number of a cograph can be computed in linear time.

Proof. The algorithm is simple and its correctness is based on Theorem 24. We now analyze its time complexity. For a cograph $G$, we can determine $G_l$ and $G_r$ in linear time by traversing its cotree. For each of $G_l$ and $G_r$, a universal, almost universal, or independent vertex can be determined in linear time by traversing the two graphs, e.g., counting the degree of every vertex is enough. Thus, the overall time complexity is linear. This proves the theorem. \(\square\)

In [2] 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 $\gamma(G) \leq \gamma(G - x)$ for every vertex $x$. A constructive characterization of Roman trees is given in [13]. In [2] it is posed as an open problem to determine Roman graphs other than trees. It would be of interest to know which cographs satisfy this equality. In Theorem 24, Statements (1) and (3) exactly characterize the Roman cographs. Together with Theorem 25, we have the following theorem.

Theorem 26. The Roman cographs can be recognized in linear time.

Proof. By Theorem 24, cographs can be classified into three classes according to their Roman domination numbers. If $G$ satisfies Statements (1) or (3) of Theorem 24, then the Roman pair $(V_1, V_2)$ of $G$ satisfies that $V_1 = \emptyset$ and $V_2$ is a minimum dominating set of $G$. If $G$ satisfies Statement (2) of Theorem 24, then the Roman pair $(V_1, V_2)$ of $G$ satisfies that $V_1 = \{u\}$ for a certain vertex $u$ and $\gamma(G) = 2$ whereas $\gamma(G - u) = 1$. Thus, $G$ is a Roman cograph if and only if $G$ satisfies Statements (1) or (3) of Theorem 24. Finally, Algorithm Roman-Cograph can be slightly modified such that if the obtained value is 2 or 4, then $G$ is a Roman cograph; otherwise $G$ is not a Roman cograph. This proves the theorem. \(\square\)

4.2. Extend to graphs of bounded cliquewidth

In this section we prove that the Roman domination problem can be expressed as a LinEMSOL($\tau_1$) optimization problem. The immediate consequence is that the Roman domination problem can be solved in linear time on any graphs $G$ having a cliquewidth bounded by a constant $k$, provided that a $k$-expression of $G$ is also a part of the input. We begin by giving some useful definitions.

Definition 27 ([8]). A $k$-expression is an expression on the vertices with labels $\{1, 2, \ldots, k\}$ of a graph using the following operations:

- $i$: create a new vertex with label $i$
- $G_1 \oplus G_2$: create a graph which is the disjoint union of a graph $G_1$ and a graph $G_2$
- $\eta_{i,j}(G)$: add all edges $uv$ in $G$ such that the label of $u$ is $i$ and the label of $v$ is $j$ ($i \neq j$)
- $\rho_{i \rightarrow j}(G)$: change the label of all vertices with label $i$ into label $j$

The cliquewidth of a graph $G$ is the minimum $k$ needed to define $G$ by a $k$-expression.
**Definition 28** *(Formal Definitions are Given in [7]).* MSOL $(\tau_1)$ denotes the monadic second order logic with quantification over subsets of vertices. Let $G$ be a graph, then $G(\tau_1)$ denotes the structure with domain $V(G)$ and binary relation $R$ such that $R(x, y) \iff xy \in E(G)$.

An optimization problem is a LinEMSOL $(\tau_1)$ optimization problem if it can be expressed as follow:

$$\arg \min_{X_1 \subseteq X : 1 \leq i \leq l} \left\{ \sum_{1 \leq i \leq l} a_i |X_i| : (G(\tau_1), X_1, \ldots, X_l) \models \theta(X_1, \ldots, X_l) \right\}$$

where $\theta$ is an MSOL($\tau_1$) formula that contains free set variables $X_1, \ldots, X_l$ and integers $a_i$ $(1 \leq i \leq l)$.

**Remark 29.** This definition is a restricted form of the one given in [7].

**Theorem 30** *(Theorem 4 in [7]).* Let $k \in \mathbb{N}$ and $C$ be a class of graphs of cliquewidth at most $k$. Then every LinEMSOL$(\tau_1)$ optimization problem on $C$ can be solved in linear time, if a $k$-expression of the graph is part of the input.

**Theorem 31.** Roman domination problem is a LinEMSOL $(\tau_1)$ optimization problem.

**Proof.** The Roman domination problem can be expressed by the following expression.

$$\arg \min_{X_1 \subseteq X : X_1 \subseteq X_2 \subseteq X} \{|X_1| + 2|X_2| : (G(\tau_1), X_1, X_2) \models \theta(X_1, X_2)\}$$

where $\theta(X_1, X_2) = \forall v (X_1(v) \lor X_2(v) \lor (\exists u (X_2(u) \land R(u, v))))$.

Clearly the formula $\theta(X_1, X_2)$ is an MSOL($\tau_1$) formula. In addition we request that any vertex must be in $X_1$, in $X_2$ or has at least one neighbor in $X_2$. Finally we search the optimal solution by the min optimization using the measure $|X_1| + 2|X_2|$.  

**Corollary 32.** The Roman domination problem can be solved in linear time on any graph $G$ with its cliquewidth bounded by a constant $k$, provided that a $k$-expression of $G$ is part of the input or there exists a linear-time algorithm to construct its $k$-expression.

**Remark 33.** In particular Roman domination problem can be solved in linear time on cographs and distance-hereditary graphs since their cliquewidths are respectively bounded by 2 and 3.

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 a $d$-octopus. Our approaches are similar to those in [18] by Kratsch, and in [11] by Fomin, Kratsch, and Müller.

We begin by providing some results on $AT$-free graphs and graphs with a $d$-octopus.

**Definition 34.** 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 35.** 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 36** *(6).* Any connected $AT$-free graph has a dominating pair.

**Definition 37.** A path $P = x_0, x_1, \ldots, x_d$ is a dominating shortest path, DSP for short, of a graph $G = (V, E)$ if
(1) $P$ is a shortest path between $x_0$ and $x_d$ in $G$, 
(2) $\{x_0, x_1, \ldots, x_d\}$ is a dominating set of $G$.

**Corollary 38 ([18])**. Every connected AT-free graph has a DSP.

**Definition 39.** 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 $x, v_1, v_2, \ldots, v_d$ of $G$, and for each $i \in \{1, \ldots, d\}$ there is a shortest path $P_i$ from $x$ 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 $x$ of the $d$ shortest paths the root of the $d$-octopus $O$. Note that the paths need not to be disjoint.

**Remark 40.** We note that the problem “Given a graph $G$ and an integer $d$, decide if $G$ has a $d$-octopus” is NP-complete (see [11]). A graph with a DSP is a graph with a 1-octopus.

The following result is a Roman domination version of Lemma 33 in [11].

**Theorem 41.** Let $G = (V, E)$ be a graph with a $d$-octopus of root $x$. Let $H_0, H_1, \ldots, H_l$ be the levels of a BFS-tree with the root $x$. Assume that $G$ has a Roman pair $(W_1, W_2)$ such that $W_2$ has at most $d$ vertices in common with each BFS-level $H_i$. Then $G$ has a Roman pair $(V_1, V_2)$ such that:

$$\sum_{i=0}^{l} \sum_{j=0}^{l-i} |V_2 \cap H_s| \leq (j + 5)d - 1. \quad (1)$$

**Proof.** The proof is similar to the proof of Lemma 33 in [11]. We start with a Roman pair $(V_{10}, V_{20})$ of $G$. If it satisfies the required property, then we are done. Otherwise, we construct a sequence of Roman pairs $(V_1, V_2), (V_2, V_3), \ldots$ 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 $Q_r = \{(i, j) : |V_{2r} \cap \bigcup_{s=i}^{i+j} H_s| \geq (j + 5)d\}$. 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 based on $i_r$, $j_r = \max\{j : (i, j) \in Q_r\}$.

Since $H_0, H_1, \ldots, H_l$ are the levels of a BFS-tree of $G$, this ensures that any neighbor of a vertex in $V_{2r} \cap \bigcup_{s=i}^{i+j} H_s$ belongs to one of the levels $H_{l-i}, H_{l-i+1}, \ldots, H_{l+j+1}$. Let $V_{l+1} = V_{1r}$ and $V_{l+2} = (V_{2r} - \bigcup_{s=i}^{i+j} H_s) + (W_2 \cap \bigcup_{s=i}^{i+j} H_s)$. Then $V_{l+2}$ dominates $V_{l+1}$. Moreover, $|V_{2r} \cap \bigcup_{s=i}^{i+j} H_s| \geq (j_r + 5)d$ and $|W_2 \cap \bigcup_{s=i}^{i+j} H_s| \leq (j_r + 5)d$, thus $|V_{2r}| \geq |V_{2r+1}|$. Therefore, $(V_{l+1}, V_{l+2})$ is also a Roman pair of $G$.

The replacement of $(V_{1r}, V_{2r})$ by $(V_{l+1}, V_{l+2})$ 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 $i_r + j_r < l$ for all steps of the construction with $Q_{r+1} \neq \emptyset$. We may assume that $i_r > 2$ and $i_r + j_r < l - 2$ since $H_s = \emptyset$ for $s < 0$ or $s > l$.

Suppose $i_{r+1} \leq i_r + j_r + 2$. The choice of $i_r$ and $j_r$ implies that $i_r + j_r + 1 \geq i_r$ since $V_{2r} \cap \bigcup_{s=0}^{i_r-3} H_s = V_{2r+1} \cap \bigcup_{s=0}^{i_r-3} H_s$. Next we have $i_{r+1} + j_r + 1 \geq i_r + j_r + 2$ since $|V_{2r+1} \cap H_{i_r}| \geq d$ for all $s$ with $i_r - 2 \leq s \leq i_r + j_r + 2$. This implies that $|V_{2r+1} \cap \bigcup_{s=i_{r+1}}^{i_{r+1}+1} H_s| = |V_{2r} \cap \bigcup_{s=i_{r+1}}^{i_{r+1}+1} H_s|$, which contradicts the selection of $i_r$ and $j_r$. Therefore, $i_{r+1} > i_r + j_r + 2$.

Although AT-free graphs are graphs with a 1-octopus, we have the following stronger result about Roman domination on AT-free graph. Our result is similar to Kratsch’s Theorem 4 in [18].
Theorem 42. Let \( G = (V, E) \) be a connected 2AT-free. 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 a BFS-tree with the root \( x \), then \( G \) has a Roman pair \( (V_1, V_2) \) such that:

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

(2)

Proof. Let \( G \) be a connected AT-free graph. There is a linear-time algorithm [17] to compute, for any given connected AT-free graph \( G \), a path \( P = (x = x_0, x_1, \ldots, x_d) \) 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 \( x_{i-1} \) or \( x_i \).

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 \( Q_r = \{(i, j) : |V_{2r} \cap \bigcup_{s=i}^{i+j} H_s| \geq j + 4\} \). 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 base on \( i_r, j_r = \max\{j : (i_r, j) \in Q_r\}\).

Every neighbor of a vertex in \( V_{2r} \cap \bigcup_{s=i}^{i+j} H_s \) belongs to one of the levels \( H_{i-1}, H_{i}, \ldots, H_{i+j+1} \). Let \( A = \{x_{i_r-1}, x_{i_r-1}, \ldots, x_{i_r+j_r+1}\} \). Since for every vertex \( z \in H_k \), \( k \in \{i_r-1, i_r, \ldots, i_r+j_r+1\} \), \( z \) is adjacent to \( x_{k-1} \) or \( x_k \), we must have \( \bigcup_{s=i}^{i+j} H_s \subseteq A \). Let \( V_{1r+1} = V_{1r} \) and \( V_{2r+1} = (V_{2r} - (\bigcup_{s=i}^{i+j} H_s)) + A \). Then \( |V_{2r} \cap (\bigcup_{s=i}^{i+j} H_s)| \geq j_r + 4 \) and \( |A| = j_r + 4 \), thus, \( |V_{2r+1}| \geq |V_{2r+1}| \). 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 \( A = \{x_{0}, x_{1}, \ldots, x_{i_r+j_r+2}\} \) contains less than \( j_r + 5 \) vertices. Therefore the Roman pair \((V_{1r+1}, V_{2r+1})\) would have smaller weight than the Roman pair \((V_{1r}, V_{2r})\), 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 \( Q_{r+1} = \{(i, j) : |V_{2r+1} \cap \bigcup_{s=i}^{i+j} H_s| \geq j + 4\} \). 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 \( j \geq i_r - 2 \) 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 \( |V_{2r+1} \cap (\bigcup_{s=i}^{i+j'} H_s)| = |V_{2r} \cap (\bigcup_{s=i}^{i+j'} H_s)| \) and thus \((i, j') \in Q_r\), contradicting the previous choice of \( i_r \) 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. \( \square \)

5.1. A polynomial-time algorithm

Our algorithm is similar to the one described in [18]. It uses dynamic programming to compute a Roman pair through the levels of a BFS-tree. A sub-solution 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 \( i - 1 \in \{1, 2, \ldots, I - 1\} \). Information of any sub-solution \( S \) that we must store during the execution are the vertices that belong to the last two current levels (i.e., \( S \cap (H_{i-2} + H_{i-1}) \)). 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 \( 7d - 1 \) for graphs with a \( d \)-octopus.

Our algorithm \( r_{p_2}(G) \), 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 \( r_{p_2}(G) \) outputs a Roman pair of \( G \).
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 non-empty subsets $S$ of $N[x]$
  - satisfying $|S| \leq k$ with $val(S) \leftarrow 2|S|$
- add to the queue $A_1$ the ordered triple $(\emptyset, \emptyset, 1)$

while $A_i \neq \emptyset$ 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 $|S + U| \leq k$ do
      - $R \leftarrow (S + U) - H_{i-2}$
      - $R' \leftarrow S' + U$
      - $val(R') \leftarrow val(S') + 2|U| + |H_{i-1} - N[S + U]|$
      - 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 $A_i$, determine one with minimum value $v = val(S') + |H_i - N[S]|$, say $(B, B', val(B'))$

if $v < val(D)$ then
  - $D \leftarrow B'$
  - $val(D) \leftarrow v$
return $(V_1, V_2) = (V - N[D], D)$

Theorem 43. 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 analysis of the running time is similar to Theorem 5 in [18]. In the following, we prove the correctness of the algorithm. For each triple $(S, S', val(S'))$, the set $S'$ represents a sub-solution corresponding to $S$ with weight $val(S')$. However, notice that the set $S'$ does not affect the running time of the algorithm and the main purpose of storing sub-solutions $S'$ 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 $(S, S', val(S'))$ in the queue $A_i$, $i \in \{1, 2, \ldots, l\}$, we have $S = S' \cap (H_{i-1} + H_i)$, $val(S') = 2|S'| + |T|$ and $\bigcup_{j=0}^{i-1} H_j \subseteq (N[S'] + T)$ with $T = (\bigcup_{j=0}^{i-1} H_j) - N[S']$. This is true for $i = 1$. By the initialization of $A_1$, for all triples $(S, S', val(S)) \in A_1$ 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 $(R, R', val(R'))$ is in $A_i$ only if there is a triple $(S, S', val(S'))$ in $A_{i-1}$ and a subset $U$ with $|S + U| \leq k$ such that $R = (S + U) - H_{i-2}$, $R' = S' + U$ and $val(R') = val(S') + 2|U| + |H_{i-1} - N[S + U]|$. Consequently, $R = R' \cap (H_{i-1} + H_i)$, $val(R') = 2|R'| + |T|$, and $(\bigcup_{j=0}^{i-1} H_j) \subseteq (N[R'] + T)$ with $T = (\bigcup_{j=0}^{i-1} H_j) - N[R']$.

Therefore, for any triple $(S, S', val(S)) \in A_l$, $(V_1, V_2) = (V - N[S'], S')$ is a Roman dominating set of $G = (V, E)$. Consequently, for any Roman pair $(V_1, V_2)$ of $G$ such that at most $k$ vertices of $V_2$ belong to any three consecutive BFS-levels of $x$. There will be a triple $(S, S', val(S'))$ in $A_l$ 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. □
Theorem 44. There is an $O(n^{7d+1})$-time algorithm to compute Roman pairs for graphs with a $d$-octopus. In particular, there is an $O(n^8)$-time algorithm to compute 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 41 and 43, $rp_{7d-1}(G)$ computes a Roman pair for a graph $G$ that is known to have a $d$-octopus. This algorithm takes time in $O(n^{7d+1})$. From the algorithm $rp_k(G)$ 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 compute a Roman pair for a graph with a DSP. Since a DSP can be compute in $O(n^3m)$ time if a graph has a DSP (see [9,18]), we can modify the algorithm $rp_k(G)$ by preprocessing the root to obtain an $O(n^7)$-time algorithm.

Analogously, $rp_5(G)$ 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 compute a dominating pair. Modifying the algorithm $rp_5(G)$ by preprocessing the dominating pair will give us an $O(n^6)$-time algorithm (see [5,18]). □

6. Conclusion

We have provided, in this paper, linear-time algorithms for the Roman domination problem on interval graphs, cographs, and graphs with bounded cliquewidth. The result for interval graph answers an open question raised in [2]. We also give a characterization of Roman cographs. It implies that Roman cographs can be recognized in line time. This also answers another open question raised in [2]. Finally, we extend the polynomial-time algorithms developed in [18] to compute Roman pairs of AT-free graphs and graphs with a $d$-octopus.

Acknowledgments

We would like to thank Dieter Kratsch and the anonymous referees for their helpful comments and advice.

References