On cocolourings and cochromatic numbers of graphs

John Gimbel
Department of Mathematics, University of Alaska – Fairbanks, Fairbanks, AK 99775-1110, USA

Dieter Kratsch
Fakultät Mathematik, Friedrich-Schiller-Universität, Universitätsstrasse, 0-6900 Jena, Germany

Lorna Stewart
Department of Computing Science, University of Alberta, Edmonton, Alta., Canada T6G 2H1

Received 28 June 1990
Revised 29 January 1992

Abstract

A cocolouring of a graph G is a partition of the vertices such that each set of the partition induces either a clique or an independent set in G. The cochromatic number of G is the smallest cardinality of a cocolouring of G. We show that the cochromatic number problem remains NP-complete for line graphs of comparability graphs (and hence for line graphs and $K_{1,s}$-free graphs), and we present polynomial time algorithms for computing the cochromatic numbers of chordal graphs and cographs.

Keywords. Cocolouring, cochromatic number, chordal graphs, cographs, graph algorithms

1. Introduction

We study only finite, simple and undirected graphs $G = (V, E)$. Throughout the paper we use $n = |V(G)|$ and $m = |E(G)|$. For $V' \subseteq V$, let $G_{V'} = (V', \{\{u, v\} \in E : u, v \in V'\})$ be the subgraph of G induced by $V'$. $I \subseteq V$ is an independent set if $\{u, v\} \notin E$ for all $u, v \in I$. $C \subseteq V$ is a clique if $\{u, v\} \in E$ for all $u, v \in C$. A colouring of the vertices of a graph $G = (V, E)$ is a partition $\{I_1, I_2, \ldots, I_r\}$ of $V$ such that for each $1 \leq j \leq r$, $I_j$ is an independent set. The chromatic number $\chi(G)$ is the minimum size of such a partition. $\kappa(G) = \chi(G)$ is the clique cover number of $G$, i.e., the minimum size of a partition.
into cliques. Analogously, a cocolouring of $G$ is a partition $\{I_1, I_2, \ldots, I_r, C_1, C_2, \ldots, C_s\}$ of $V$ such that each $I_j$, $1 \leq j \leq r$, is an independent set and each $C_j$, $1 \leq j \leq s$, is a clique. The smallest cardinality of a cocolouring of $G$ is the cochromatic number $z(G)$. Therefore, $z(G) \leq \min \{\chi(G), \kappa(G)\}$.

The cochromatic number was originally studied in [14]. Subsequent papers addressed various topics including the structure of critical graphs, and bounds on the cochromatic numbers of graphs with certain properties (e.g., fixed number of vertices, bounded clique size, fixed genus) [6, 9–11, 16, 17]. However, there is another motivation: if one has to sort a repetition-free sequence of integers (one may assume a permutation of the first $n$ integers), it is desirable to have a partition into a small number of sets of already sorted elements, i.e., subsequences which are either increasing or decreasing. This problem was studied in [1], and in [18], Wagner showed that the problem of deciding whether a given sequence can be partitioned into $k$ increasing or decreasing subsequences is NP-complete, if $k$ is a part of the input. Now the minimum $k$ is exactly the cochromatic number of the corresponding permutation graph. Furthermore, it corresponds to the partition of a poset of dimension 2 into $k$ chains or antichains, $k$ part of the input.

Given a graph $G$ and two integers $r$ and $s$, the problem of determining whether there exists a cocolouring of $G$ with $r$ independent sets and $s$ cliques (COCOLOURING) is obviously NP-complete since it contains both GRAPH-$k$-COLORABILITY (GT4 of [7]) and PARTITION INTO CLIQUES (GT15 of [7]) as special cases. Of course, the COCOLOURING problem remains NP-complete for any family of graphs where either of the special cases is NP-complete, including planar graphs, circular arc graphs, circle graphs, line graphs, and $K_{1,3}$-free graphs (see [13]). In addition, the similar COCHROMATIC NUMBER problem, given a graph $G$ and an integer $k$, determine whether $z(G) \leq k$, has been shown to be NP-complete for permutation graphs [18], and hence it remains NP-complete for circle graphs. For graph classes not defined here we refer to [12, 13].

In Section 2 we show that COCHROMATIC NUMBER remains NP-complete for line graphs of comparability graphs, and hence for line graphs and $K_{1,3}$-free graphs. Then, we present polynomial time algorithms for computing the cochromatic numbers of chordal graphs and cographs. Both of these algorithms check whether there is an $r,s$-cocolouring for different values of $r$ and $s$ by computing the minimum $s$ for any fixed $r$, $0 \leq r \leq \chi(G)$, or the minimum $r$ for any fixed $s$, $0 \leq s \leq \kappa(G)$. Hence the minimum $z := r + s$ over all $r$, $s$ such that $G$ has an $r,s$-cocolouring is the cochromatic number of the graph.

2. NP-completeness results

We use a single transformation to show that COCHROMATIC NUMBER remains NP-complete when restricted to certain graph classes.
Lemma 2.1. (1) $\chi(G) = z(nG)$, where $nG$ is the disjoint union of $n$ copies of $G$.

(2) $\text{GRAPH-}k\text{-COLORABILITY } (\mathcal{G}_1) \approx \text{COCHROMATIC NUMBER } (\mathcal{G}_2)$, where $\mathcal{G}_2 = \{ nG : G \in \mathcal{G}_1, n = |V(G)| \}$.

(3) If $\mathcal{G}$ is a graph class closed under taking disjoint union then the NP-completeness of $\text{GRAPH-}k\text{-COLORABILITY}$ on $\mathcal{G}$ implies the NP-completeness of $\text{COCHROMATIC NUMBER}$ on $\mathcal{G}$.

(4) If $\mathcal{G}$ is a graph class closed under taking disjoint union then the NP-completeness of $\text{GRAPH-}k\text{-COLORABILITY}$ for fixed $k$ on $\mathcal{G}$ implies the NP-completeness of $\text{COCHROMATIC NUMBER}$ for the same fixed $k$ on $\mathcal{G}$.

Proof. (1) Obviously, $z(nG) \leq \chi(nG) = \chi(G)$. Assume we have $s \geq 1$ cliques $C_1, \ldots, C_s$ in an optimal cocolouring of $nG$. Then $nG - \bigcup_{i=1}^s C_i$ contains $(n - s)G$ as an induced subgraph. Therefore we have at least $\chi(G)$ independent sets in the cocolouring; thus such a cocolouring is not optimal.

(2) The transformation is to construct for $G \in \mathcal{G}_1$ the graph $nG \in \mathcal{G}_2$. Then, by (1), we have:

$$\chi(G) \leq k \iff z(nG) \leq k.$$

(3) and (4) are obvious consequences of (2). \(\square\)

Corollary 2.2. (1) $\text{COCHROMATIC NUMBER}$ remains NP-complete on circle graphs, line graphs of comparability graphs, and hence for line graphs and $K_{1,3}$-free graphs.

(2) $\text{COCHROMATIC NUMBER}$ remains NP-complete for any fixed $k \geq 3$ on planar graphs with no vertex degree exceeding 4.

Proof. First, all these classes of graphs are closed under taking disjoint union. Thus, by (3) and (4), it is enough to show that $\text{GRAPH-}k\text{-COLORABILITY}$ remains NP-complete for that class, with the same restriction on $k$. For all except line graphs of comparability graphs this is mentioned in [7, 13].

$\text{GRAPH-}k\text{-COLORABILITY}$ is NP-complete on line graphs of comparability graphs since $\text{CHROMATIC INDEX} = \min \{ (G, k) : \text{there is an edge colouring of } G \text{ with at most } k \text{ colours such that adjacent edges have different colours} \}$ remains NP-complete on comparability graphs [2]. \(\square\)

Unfortunately circular arc graphs are not closed under disjoint union. We leave the complexity of $\text{COCHROMATIC NUMBER}$ on circular arc graphs as an open problem.

3. Computing the cochromatic number of a chordal graph

A graph $G = (V, E)$ is chordal if every cycle of length exceeding 3 has a chord, i.e., an edge joining two nonconsecutive vertices in the cycle. The property we will need is
that chordal graphs are exactly the intersection graphs of subtrees in a tree [8]. More
exactly, for each chordal graph $G = (V, E)$ there exists a tree $T$ such that

- the vertices of $T$ correspond to the maximal cliques of $G$, and
- the vertices of $T$ corresponding to the cliques of $G$ containing any fixed vertex
  $v \in V$ form a subtree of $T$. This subtree is said to correspond to the vertex $v$.

Note the consequence that two vertices of $G$ are adjacent if and only if their
respective subtrees have nonempty vertex intersection. For a given chordal graph
$G = (V, E)$, such a tree, called a clique tree for $G$, will have no more than $n$ nodes, and
can be constructed in $O(n + m)$ time [15].

3.1. The algorithm

We now describe an algorithm for computing the cochromatic number of a chordal
graph. In a $k$-cocolouring of any graph, the graph is partitioned into $r \geq 0$ indepen-
dent sets and $k - r \geq 0$ cliques. Our algorithm for chordal graphs uses the following
strategy. For each $0 \leq r \leq \chi(G)$, we calculate the minimum cocolouring number for
the input graph $G$ which can be realized by a partition containing $\leq r$ independent
sets. We then take the minimum over all $0 \leq r \leq \chi(G)$. It is possible to compute
the minimum cocolouring number for each $r$ in polynomial time for a chordal
graph $G$, using a pruning order traversal of a clique tree for $G$. A pruning
order traversal of a tree is one in which all neighbours of a node, except at most
one must be visited before the node itself is visited. That is, the next node to be visited
is a leaf in the subtree consisting only of unvisited nodes. The algorithm traverses
the clique tree according to a pruning order, which remains fixed throughout
the computation for a particular $r$-value. The currently traversed node is included
as a clique in the cocolouring if and only if a particular subgraph of $G$ is not
$r$-colourable.

Let $T$ be a clique tree for a chordal graph $G$. The algorithm uses the following data
structures and notation to represent the necessary information. $T$ is a tree with the
usual pointers which enable a pruning order traversal. We will use $C_i$ to denote both
a node in the tree and the corresponding maximal clique in $G$. Let $C_i$ be the node of
$T$ which is the current node in a pruning order traversal. $T_{C_i}$ will denote the subtree of
$T$ consisting of $C_i$ and all nodes of $T$ connected to $C_i$ by paths consisting entirely of
previously visited vertices. $G_{C_i}$ will denote the subgraph of $G$ which is induced by all
the vertices occurring in cliques corresponding to nodes of $T_{C_i}$. $G_{C_i} - C_j$ refers to the
subgraph of $G_{C_i}$ obtained by removing all vertices of $C_j$ from $G_{C_i}$. The algorithm may
now be stated as follows.

Algorithm 1.

$z := \chi(G)$

For $r := 0$ to $\chi(G)$ do

$s := 0$
For each node $C_i$ of $T$ in a pruning order do
\{ Let $C_j$ be the neighbour of $C_i$ in $T$ which has not yet been visited.\} 
\{ If no such neighbour exists then $C_i$ is the last node of $T$ to be visited; \} 
\{ in this case, we assume $C_j = \emptyset$. \} 
If $G_{C_i} - C_j$ is not $r$-colourable then
\quad $G := G - G_{C_i}$
\quad Update $T$ to represent $G - G_{C_i}$
\quad $s := s + 1$
End
$z := \min \{ z, s + r \}$
Reset $T$ and $G$
End

We will examine the execution of Algorithm 1 on the chordal graph $G$ and its clique tree $T$ of Fig. 1. Table 1 shows the cliques triggering subtree deletions and the values of $s$ during the execution of the algorithm on $G$ and $T$. In this example, the algorithm terminates with the result: $z(G) = 3$. The algorithm’s choice of $C_3$ and $C_4$ for $r = 1$ (i.e., the $r$-value for which $r + s$ is minimized) means that there is an optimal cocolouring consisting of the two cliques, $\{2, 4, 5, 6, 7\}$ and $\{8, 9, 10, 11\}$, and one independent set. Table 2 shows the algorithm’s execution in detail for $r = 2$. Initially, $s = 0$, and $G$ and $T$ are as shown in Fig. 1. Subsequent values of $G$ and $T$ are shown in Fig. 2. The final value of $s = 2$ means that any cocolouring of $G$ with two independent sets must have at least two cliques, and that there is a cocolouring of $G$ consisting of two independent

![Chordal graph, G:](image)

![Clique tree, T, with a pruning order indicated:](image)

Fig. 1. A chordal graph $G$ and its clique tree $T$. 
Table 1
Execution of Algorithm 1 on G and T

<table>
<thead>
<tr>
<th>r</th>
<th>cliques triggering subtree deletions</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$C_1, C_2, C_3, C_5, C_6, C_7, C_8$</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>$C_3, C_4$</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>$C_4, C_6$</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>$C_6$</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$C_8$</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>none</td>
<td>0</td>
</tr>
</tbody>
</table>

(subscripts refer to the pruning order of Fig. 1)

Table 2
Execution of Algorithm 1 on G and T: details for $r = 2$

<table>
<thead>
<tr>
<th>i</th>
<th>vertices of $G_C$</th>
<th>vertices of $G_{C_i}$ – $C_j$</th>
<th>2-colourable?</th>
<th>$G$</th>
<th>$T$</th>
<th>$s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1, 2]</td>
<td>[1]</td>
<td>yes</td>
<td>$G$ of Fig. 1</td>
<td>$T$ of Fig. 1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>[3, 4]</td>
<td>[3]</td>
<td>yes</td>
<td>$G$ of Fig. 1</td>
<td>$T$ of Fig. 1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>[1, 2, 3, 4, 5, 6, 7]</td>
<td>[1, 2, 3, 4]</td>
<td>yes</td>
<td>$G$ of Fig. 1</td>
<td>$T$ of Fig. 1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>[8, 9, 10, 11]</td>
<td>[9, 10, 11]</td>
<td>no</td>
<td>$G_1$ of Fig. 2</td>
<td>$T_1$ of Fig. 2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>[14]</td>
<td>[14]</td>
<td>yes</td>
<td>$G_1$ of Fig. 2</td>
<td>$T_1$ of Fig. 2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>[1, 2, 3, 4, 5, 6, 7, 14]</td>
<td>[1, 2, 3, 4, 5, 6, 7, 14]</td>
<td>no</td>
<td>$G_2$ of Fig. 2</td>
<td>$T_2$ of Fig. 2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>[12]</td>
<td>[12]</td>
<td>yes</td>
<td>$G_2$ of Fig. 2</td>
<td>$T_2$ of Fig. 2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>[12, 13]</td>
<td>[12, 13]</td>
<td>yes</td>
<td>$G_2$ of Fig. 2</td>
<td>$T_2$ of Fig. 2</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 2. Execution of Algorithm 1 on G and T: values of G and T for $r = 2$. 
sets and two cliques. Furthermore, the execution of the algorithm shows that there is such a cocolouring containing the cliques \{8, 9, 10, 11\} and \{5, 6, 7\}.

We now turn our attention to the correctness of Algorithm 1.

**Theorem 3.1.** Algorithm 1 correctly calculates the cochromatic number for any input chordal graph \(G\).

**Proof.** We have to show that for each \(r\) the algorithm produces the smallest \(s\) such that the graph can be partitioned into \(r\) independent sets and \(s\) cliques. If this is the case, the value of \(z\) after termination is exactly \(z(G)\).

Assume that for some \(r\) the value \(s\) returned by the algorithm is not optimal. Let \(\{C_1, C_2, \ldots, C_s\}\) be the collection of maximal cliques triggering the deletions of subtrees during the execution of the algorithm on \(G\) for our value \(r\), in the order of their deletion. (Note that these are exactly the roots of the deleted subtrees, if we consider the tree to be rooted according to the parsing order used.) Let \(C_1, C_2, \ldots, C_{s'}, s' < s\) be a sequence of cliques whose removal from \(G\) results in an \(r\)-colourable graph. Suppose this sequence is chosen such that the index, say \(k\), at which \(C_1, C_2, \ldots, C_s\) and \(C_1, C_2, \ldots, C_{s'}\) first differ is as large as possible. (Note that the sequence \(C_1, C_2, \ldots, C_{s'}\) does not necessarily correspond to any execution of the algorithm.)

Let \(G_{k-1}^{k-1}\) be the subgraph of the input which remains after the removal of the vertices \(\bigcup_{i=1}^{k-1} V(G_{C_i})\) from \(G\), i.e., \(G^{k-1}\) is the value of \(G\) after the \((k-1)\)st subgraph deletion in the algorithm. Let \(T^{k-1}\) be the corresponding value of \(T\) in the algorithm. Since the next subgraph deleted by the algorithm is \(G_{C_k}^{k-1}\) (by the definitions of \(C_1, C_2, \ldots, C_s\) and \(k\)), we know that the induced subgraph \(G_{C_k}^{k-1} - C_j\) of \(G^{k-1}\) (where \(C_j\) is the unvisited neighbour of \(C_k\) in \(T^{k-1}\)) is not \(r\)-colourable. Now any vertex of \(G_{C_k}^{k-1}\) which occurs in a tree node outside of \(T^{k-1}\) must occur in \(C_j\) since otherwise the subtree for that vertex would be disconnected. Thus, the only cliques of \(G_{C_k}^{k-1}\) whose removal can affect the colourability of \(G_{C_k}^{k-1} - C_j\) are those corresponding to nodes of \(T_{C_k}^{k-1}\). So \(C_1, C_{k+1}, \ldots, C_{s'}\), must contain a clique, say \(C_{i}'\), of the subtree \(T_{C_k}^{k-1}\). But then \(G^{k-1} - G_{C_i}'\) is a subgraph of \(G^{k-1} - G_{C_k}\); hence \(\chi(G^{k-1} - G_{C_i}') \leq \chi(G^{k-1} - G_{C_k}).\) Thus, since \(C_1, C_2, \ldots, C_{k-1}, \ldots, C_{i}'\ldots, C_{r'}\) is a cocolouring for \(G\) with \(r\) independent sets, \(C_1, C_2, \ldots, C_{k-1}, C_k, \ldots, C_{i+1}, \ldots, C_{r'}\) (i.e., the sequence obtained from \(C_1, C_2, \ldots, C_{r'}\) by deleting \(C_i\) and inserting \(C_k\) after \(C_{k-1}\)) is a cocolouring for \(G\) with \(r\) independent sets and \(s'\) cliques which first differs from the one produced by the algorithm at a larger index. This contradicts our choice of \(C_1, C_2, \ldots, C_{s'}\). We conclude that the algorithm correctly calculates the cochromatic number of \(G\). \(\square\)

3.2. Implementation details and timing analysis

We now describe an efficient implementation of Algorithm 1 and analyze its time complexity.
Lemma 3.2. \( \chi(G_{C_i} - C_j) = \max_{nodes C_k of T_{c_i}} |C_k - C_j| \).

Proof. Since \( G_{C_i} - C_j \) is an induced subgraph of a chordal graph \( G \), it is chordal and therefore perfect. This tells us that \( \chi(G_{C_i} - C_j) \) is equal to the size of a maximum clique in \( G_{C_i} - C_j \) (see [12]). Now, \( G_{C_i} - C_j \) is the subgraph of \( G_{C_i} \) induced by vertices which occur only in cliques of \( T_{C_i} \). Thus, \( T_{C_i} \), with vertices of \( C_j \) removed, is a tree for \( G_{C_i} - C_j \), the nodes of which include all of the maximal cliques of \( G_{C_i} - C_j \). Furthermore, \( T_{C_i} \), with vertices of \( C_j \) removed, has all of the properties of a clique tree, except that it may contain some nodes corresponding to cliques which are not maximal. The lemma follows. \( \square \)

Thus, one approach to determining whether \( G_{C_i} - C_j \) is \( r \)-colourable would be to scan the nodes \( C_k \) of \( T_{C_i} \), checking \( |C_k - C_j| \) for each one. However, we can check \( r \)-colourability more efficiently as follows. We store an indication, \( count \), of the colourability of each clique in its corresponding tree node. Before a traversal begins, \( count \) is set to zero for all nodes. Then, at each step of the pruning order traversal, for each \( C_k \) in \( T_{C_i} \), \( count(C_k) \) is incremented by \( |C_k \cap (C_i - C_j)| \). If any \( count \) value becomes greater than \( r \), then the entire \( T_{C_i} \) subtree is removed from \( T \), and vertices occurring in \( T_{C_i} \) are removed from all cliques of \( T - T_{C_i} \) in which they occur, as well. This is accomplished efficiently with the aid of a list, for each \( v \in V \), of pointers to the tree nodes containing \( v \), and, for each clique in \( T \), a bit string of length \( n \) in which the \( i \)th bit is a 1 if and only if vertex \( i \) is in the corresponding clique. The entire algorithm, including implementation details, follows.

Algorithm 2.

I

For each node \( C_i \) of \( T \) do

1. Set the bit string of \( C_i \) to all zeros

2. For each \( v \in C_i \) do

   Add to \( v \)'s list a pointer to \( C_i \) in \( T \)
   In the bit string of \( C_i \), change \( v \)'s bit to one

End

count \( (C_i) := 0 \)

End

End

\( z := \chi(G) \)

For \( r := 0 \) to \( \chi(G) \) do

\( s := 0 \)

II

For each node \( C_i \) of \( T \) in a pruning order do

1. \{ Let \( C_j \) be the neighbour of \( C_i \) in \( T \) which has not yet been visited.\}
2. \{ If no such neighbour exists then \( C_i \) is the last node of \( T \) to be visited;\}
3. \{ in this case, we assume \( C_j = \emptyset \)\}

\( r \)-colourable := true
Cocolourings and cochromatic numbers

III

For each \( v \in C_i - C_j \) do

For each node \( C_k \) of \( T_{C_i} \) containing \( v \) do

\[
\text{count}(C_k) := \text{count}(C_k) + 1
\]

If \( \text{count}(C_k) > r \) then

\( r\text{-colourable} := \text{false} \)

End

End

If not \( r\text{-colourable} \) then

\( T := T - T_{C_i} \)

IV

For each \( v \in C_i \cap C_j \) do

For each clique \( C \) containing \( v \) do

Delete \( v \) from \( C \) by changing the appropriate bit to zero

End

End

\( s := s + 1 \)

End

\( z := \min \{z, s + r\} \)

Reset \( T \) and \( G \), including the initialization of loop I

End

We must be sure that Algorithm 2 is a correct implementation of Algorithm 1.

Lemma 3.3. Upon termination of loop III, \( \text{count}(C_k) = |C_k - C_j| \) for all \( C_k \) in \( T_{C_i} \).

Proof. We prove this by induction on the number of nodes in \( T_{C_i} \). Notice that \( \text{count}(C_i) = 0 \) upon entry to loop III, since only previously visited nodes have \( \text{count} \neq 0 \). If there is only one node in \( T_{C_i} \) then \( C_i \) is a leaf of \( T \), and the algorithm correctly computes \( \text{count}(C_i) = |C_i - C_j| \). Suppose the equality holds for all \( T_{C_i} \) sub-trees with \(< p \) nodes, and suppose the current \( T_{C_i} \) has \( p \) nodes. For each \( C_k \) of \( T_{C_i} \), the algorithm computes

\[
\text{count}(C_k) = \text{count}(C_k) + |(C_i - C_j) \cap C_k|.
\]

Upon entry to the loop, the value of \( \text{count}(C_k) \) was \( |C_k - C_i| \) by induction, and by the traversal order. This is because in a previous step of the traversal, the neighbour of \( C_i \) on the \( C_i - C_k \) path was the current node. Thus, upon termination of loop III, we have

\[
\text{count}(C_k) = |C_k - C_i| + |(C_i - C_j) \cap C_k|
\]

\[
= |C_k| - |C_i \cap C_k| + |C_i \cap C_k| - |C_j \cap C_k|
\]

\[
= |C_k| - |C_j \cap C_k|
\]

\[
= |C_k - C_j|. \quad \square
\]
Theorem 3.4. Algorithm 2 correctly computes \( z \), the cochromatic number of the input chordal graph.

Proof. We consider one iteration of loop II. Suppose that on entry to loop II, \( T \) is a clique tree for a chordal graph \( G \). We need to prove the following.

- \( G_c - C_i \) is \( r \)-colourable if and only if the variable \( r \)-colourable has the value true upon termination of loop III. This follows from Lemmas 3.2 and 3.3.
- Upon termination of loop IV, \( T \) is a tree for \( G - G_c \), with all of the properties of a clique tree except that it may contain some nodes corresponding to cliques which are not maximal. In order to change \( T \) from such a tree for \( G \) to such a tree for \( G - G_c \), all vertices of \( G_c \) must be removed from cliques of \( T \). Nodes of \( T_c \) consist entirely of such vertices; therefore all nodes of \( T_c \) must be removed. The remaining vertices of \( G_c \) are those which appear in \( T_c \), and outside of \( T_c \). All such vertices must appear in \( C_i \) and in \( C_j \), since otherwise the subtree corresponding to the vertex would be disconnected. Thus, the algorithm correctly updates \( T \). \( \square \)

Theorem 3.5. The time complexity of Algorithm 2 is \( O(n \cdot (n + m)) \).

Proof. Loop I is executed in linear time, since the sum of the clique size over all maximal cliques is \( O(n + m) \) (see [12]). What is the time complexity of one traversal over all nodes of the clique tree, i.e., loop II? For a particular \( C_i, C_j \) pair, loop III requires \( \sum_{v \in C_i - C_j} \) (number of cliques containing \( v \)) steps. For each vertex \( v \), there is exactly one \( i, j \) pair such that \( v \in C_i - C_j \). This is a consequence of the traversal order and the fact that cliques containing a vertex form a connected subtree. Thus, the total time for loop III for all nodes of \( T \) is \( O(\sum_{v \in C_i - C_j} \text{number of cliques containing } v) = O(n + m) \). Loop IV clearly executes in \( O(\sum_{v \in C_i} \text{number of cliques containing } v) \) for the current node \( C_i \), which is \( O(n + m) \) in total. Now, given a chordal graph \( G \), a clique tree for \( G \) can be constructed in \( O(n + m) \) time and \( \chi(G) \) can be computed in \( O(n + m) \) time. Thus, the overall time complexity of the algorithm is \( O(\chi(G) \cdot (n + m)) = O(n \cdot (n + m)) \). \( \square \)

4. Computing the cochromatic number of a cograph

Cographs are the graphs formed from a single vertex under the closure of the operations of union and complement. Equivalently, they are precisely those graphs containing no \( P_4 \), i.e., no chordless path on four vertices, as an induced subgraph [3].

It is known that cographs have a unique tree representation, called a cotree [3], and that the unique tree representation for a given cograph can be constructed in linear time [5]. Leaves of the tree correspond to vertices of the graph, and each internal node is either a 0-node or a 1-node. Along any leaf-to-root path, 0-nodes and 1-nodes alternate. Vertices \( x \) and \( y \) are adjacent in the cograph if and only if the path from leaf \( x \) to the root and the path from leaf \( y \) to the root first meet at a 1-node in the cotree.
A number of problems can be solved efficiently for cographs by making use of the cotree representation. Typically, these algorithms involve computing intermediate results for each induced subgraph corresponding to an internal node of the tree. The subgraph corresponding to an internal node is the cograph which is represented by the subtree rooted at that node. During a postorder traversal of the tree, the algorithm computes a solution for the subgraph corresponding to the current node from the previously computed solutions for subgraphs corresponding to children of the current node. If the current node is a 0-node, this means combining solutions for the connected components of a graph to obtain a solution for the disconnected graph as a whole. If the current node is a 1-node, we must combine solutions for subgraphs which are completely connected to one another. Polynomial cograph algorithms of this type include algorithms for the following problems: maximum clique, maximum independent set, vertex colouring, minimum clique cover, clustering, domination, minimum fill-in, and Hamiltonicity [335]. Linear time recognition and isomorphism algorithms follow from the cotree construction algorithm.

4.1. The algorithm

We present Algorithm 3, an O(n^2) algorithm which uses the cotree representation in a way similar to the algorithms in [3]. As in that paper, we specify our algorithm by appropriate substitutions for the leaves, 0-nodes and 1-nodes. To every vertex of the cotree we assign a list of pairs \([a_i, b_i]\), indicating all optimal cocolourings of the corresponding induced subgraph of \(G\); \(a_i\)'s represent numbers of independent sets and \(b_i\)'s represent numbers of cliques in a cocolouring. To any leaf of the cotree is assigned a list

\[
\emptyset = \begin{pmatrix} [1, 0] \\ [0, 1] \end{pmatrix}.
\]

To any 1-node, corresponding to a subgraph \(G'\) of \(G\), is assigned a list

\[
\emptyset = \begin{pmatrix} [a_0, s] \\ [a_1, s - 1] \\ \vdots \\ [a_{s-1}, 1] \\ [a_s, 0] \end{pmatrix}
\]

where \(s = \kappa(G')\) and \(a_0 = 0\). For every element \([a_{s-i}, i]\), \(0 \leq i \leq s\), we want the property that in any cocolouring of \(G'\) with (at most) \(i\) cliques the number of independent sets is at least \(a_{s-i}\), and \(a_{s-i}\) is possible.
To any 0-node, corresponding to a subgraph $G'$ of $G$, is assigned a list

$$
\mathcal{B} = \begin{pmatrix}
[0, b_0] \\
[1, b_1] \\
\vdots \\
[r-1, b_{r-1}] \\
[r, b_r]
\end{pmatrix}
$$

where $r = \chi(G')$ and $b_0 = 0$. For every element $[i, b_i]$, with $0 \leq i \leq r$, we want $b_i$ to be the smallest number of cliques in any cocolouring of $G'$ with (at most) $i$ independent sets. (Note that we can restrict our attention to exactly $i$ cliques and exactly $i$ independent sets, respectively, since monotonicity implies that the minimum values for $a_i$ and $b_i$, respectively, are attained in these cases.)

If these lists are correctly computed for any node of the cotree, then it is clear that the list of the root allows us to compute the cochromatic number of $G$. The algorithm computes the minimum over all sums $a_i + b_i$ for elements in the list of the root and outputs this value as $z(G)$.

We have to show how the elements of a list can be computed from the (correctly computed) lists for the children of this node. Thereby our aim is to make sure that the following condition is fulfilled, a reformulation of our requirements for the elements of a list, given in the definitions of the lists.

A cocolouring of $G'$ with $a_i$ independent sets contains at least $b_i$ cliques (for 0-nodes and leaves), a cocolouring of $G'$ with $b_i$ cliques contains at least $a_i$ independent sets (for 1-nodes and leaves), and there is a cocolouring of $G'$ with $a_i$ independent sets and $b_i$ cliques. (*)

Therefore, the minimum over $a_i + b_i$ for all $[a_i, b_i]$ in $\mathcal{B}$ would be $z(G')$, if $G'$ is the subgraph corresponding to the list. This would prove the correctness of the algorithm, by considering the list of the root of the cotree and the corresponding graph $G' = G$.

Let $G'$ be the subgraph corresponding to an internal node and let $G_1, G_2, \ldots, G_k$ be the subgraphs corresponding to its children. We start with the following obvious formulas:

If $G' = \overline{G_1 \cup G_2 \cup \cdots \cup G_k}$, i.e., $G'$ corresponds to a 1-node and is the join of $G_1, G_2, \ldots, G_k$ then

$$
\kappa(G') = \kappa(\overline{G_1 \cup G_2 \cup \cdots \cup G_k}) = \max_{1 \leq i \leq k} \kappa(G_i),
$$

$$
\chi(G') = \chi(\overline{G_1 \cup G_2 \cup \cdots \cup G_k}) = \sum_{i=1}^{k} \chi(G_i).
$$

(1)
Cocolourings and cochromatic numbers

If \( G' = G_1 \cup G_2 \cup \cdots \cup G_k \), i.e., \( G' \) corresponds to a 0-node and \( G' \) is the disjoint union of \( G_1, G_2, \ldots, G_k \) then

\[
\chi(G') = \chi(G_1 \cup G_2 \cup \cdots \cup G_k) = \max_{1 \leq i \leq k} \chi(G_i),
\]

(2)

\[
\kappa(G') = \kappa(G_1 \cup G_2 \cup \cdots \cup G_k) = \sum_{i=1}^{k} \kappa(G_i).
\]

Now we give the proof of (*) by induction on the height of the nodes in the cotree. The lists of the leaves obviously fulfill (*). Now let \( G' \) be an induced subgraph of \( G \) corresponding to an internal node of the cotree. Let \( G_1, G_2, \ldots, G_k \) correspond to the children of this node and let the assigned lists \( \vartheta_1, \vartheta_2, \ldots, \vartheta_k \) be already correctly computed. Hence, (*) is fulfilled for these lists. For the induction on the height of the nodes, we have only to distinguish whether \( G' \) corresponds to a 0-node or to a 1-node. (To avoid confusion, we label elements of a list \( \vartheta_v \) with a superscript \( v \), writing e.g. \( b_{ij}^{v} \) for the \( b_{ij} \)-entry of list \( \vartheta_v \).

Case 1: \( G' \) corresponds to a 1-node of the cotree.

For computing \( a_{i-j} \), \( 0 \leq j \leq s \), we have to fix the number of cliques in the cocolouring of \( G' \) to be \( j \). By (1), the number of independent sets in a cocolouring of \( G' \) is equal to the sum of the independent sets induced by this cocolouring in the subgraphs \( G_v, 1 \leq v \leq k \), and the number of cliques is equal to the maximum number of cliques in \( G_v, 1 \leq v \leq k \). Hence for finding the minimum number of independent sets in a cocolouring with \( j \) cliques, we have to choose \( j \) cliques in every \( G_v \), and look for the minimum number of independent sets we need there. Since the assigned lists are correctly computed, and by (*), we know that choosing \( j \) cliques in some \( G_v \) corresponds to choosing the smallest \( l_v \) with \( b_{ij}^v = j \leq l_v \). (Note that (*) implies that for any \( a_{ij}^v \) with \( b_{ij}^v > j \), more than \( j \) cliques have to be chosen in any corresponding cocolouring of \( G' \). Furthermore, the \( b_{ij}^v \)'s are decreasing in every list \( \vartheta_v \), thus the smallest possible \( l_v \) leads to the largest number \( (\leq j) \) of cliques in \( G_v \), namely \( b_{ij}^v \).)

Therefore, we have:

\[
\text{For every } j \in \{0, 1, \ldots, s\}, \ a_{i-j} = \sum_{v=1}^{k} a_{i-j}^v,
\]

where, for every \( v \in \{1, 2, \ldots, k\}, \ b_{i-j}^v \leq j \) and \( (b_{i-j}^{v-1} > j \text{ or } l_v = 0) \). (3)

Thus we have a good rule for computing the list for any 1-node, if the lists for all its children are computed.

Case 2: \( G' \) corresponds to a 0-node of the cotree.

For computing \( b_{ij} \), \( 0 \leq j \leq r \), we have to fix the number of independent sets in the considered cocolourings of \( G' \) to be \( j \). To minimize the number of cliques in such a cocolouring of \( G' \), by (2), we have to choose \( j \) independent sets in the induced cocolourings of the subgraphs \( G_v, 1 \leq v \leq k \). Similar to Case 1, since the lists of the
children are correctly computed and (*) is fulfilled for all of them, we cannot choose a \( b'_m \) with \( a'_m^v > j \) for some \( v \in \{1, 2, \ldots, k\} \). Any cocolouring with \( b'_m \) cliques in \( G_v \) would have at least \( a'_m^v > j \) independent sets on \( G' \). On the other hand, the maximum \( m_v \) with \( a'_m = j' < j \) minimizes the corresponding \( b'_m \) for every \( G_v \), hence minimizing the number of cliques for the corresponding cocolouring of \( G' \).

For every \( j \in \{0, 1, \ldots, r\} \), \( b_j = \sum_{v=1}^{k} b'_m^v \),

where, for every \( v \in \{1, 2, \ldots, k\} \), \( a'_m^v \leq j \) and \( (a'_m^{v+1} > j \) or \( b'_m^v = 0 \). (4)

Consequently, Algorithm 3 computes the lists for the nodes of the cotree from the bottom to the top, by a postorder traversal of the cotree, according to (3) and (4), and computes the cochromatic number from the list of the root, which corresponds to the given graph \( G \).

An example of the execution of Algorithm 3 is presented in Fig. 3.

From our considerations above, we have:

**Theorem 4.1.** Algorithm 3 correctly computes the cochromatic number of a cograph.

### 4.2. Implementation details and timing analysis

We show how to get the values \( a'_m^v \) and \( b'_m^v \) efficiently from the lists of the children.

**Theorem 4.2.** Algorithm 3 can be implemented to run in time \( O(n^2) \).

**Proof.** The postorder traversal of the tree needs \( O(n) \) time. The list for a node of the cotree is stored as a doubly linked list where every element is an array with the two entries \( a_i, b_i \). The order in the linked list is the order of the indices, i.e., the first element is \([a_0, b_0]\), the second is \([a_1, b_1]\), and so on. For an efficient implementation it is convenient to compute any new list exactly in this order. Now computing a new list for a 1-node (respectively 0-node) from the lists of its children is done by scanning a pointer through the lists of the children, starting from the last (respectively first) element and moving to the first (respectively last) one. The pointers are not repositioned; they start for getting the next value where they stopped for the preceding one. Hence any list is scanned at most once, during the computation of the list for its parent in the cotree. The scanning rules follow from (3) and (4):

1-node: If we compute \( a_j \geq a_{j-1} \), its predecessor, we have to find, in every \( \beta_v, a'_v \). The pointer starts where it stopped for computing \( a_{j-1} \) and stops on the last \( b'_m \)-entry smaller than or equal to \( j \), getting \( b'_m \leq j \), or on the first element of the list. Since the \( b'_m \) are decreasing with respect to \( i \), we have \( b_{i-1} > j \) or \( i = 0 \). Therefore, summing up the values indicated by the pointers gives the correct value of \( a_{i-j} \), according to (3).
Fig. 3. A cograph $G$, its cotree and the result of Algorithm 3.

0-node: If we compute $b_j$, looking for $b_j^c$ in list $\mathcal{E}$, for every list $\mathcal{E}_v$ of the children, then the pointer starts, where it stops for the value of $b_{j-1}^c$, and scans through the list up to the predecessor of the first element $[a^c_{m_v+1}, b^c_{m_v+1}]$ with $a^c_{m_v+1} > j$, or stops on the last entry of the list. Since the $a^c_m$ are increasing with respect to $m$, we have $a^c_m \leq j$, and either $a^c_{m_v+1} > j$ or $a^c_m$ is the last element of the list. Therefore, we correctly compute the $b_j$, according to (4).

Finally, computing a list requires a scan through the list for all of the children. Hence, any list is scanned at most once during the whole computation. Any list has $O(n)$ elements, thus the algorithm has running time $O(n^2)$. □
5. Conclusions

We have shown that COCHROMATIC NUMBER is NP-complete for line graphs of comparability graphs (and therefore for line graphs and \(K_{1,3}\)-free graphs), and we have presented polynomial time algorithms for computing the cochromatic numbers of chordal graphs and cographs. Our algorithms imply that the problem of partitioning a partial order into a minimum number of chains or antichains is solvable in polynomial time for cycle free orders, the class of posets whose comparability graphs are chordal, for series parallel orders, the class of posets whose comparability graphs are cographs, and for interval orders, the class of posets whose comparability graphs are the complements of interval graphs.

As mentioned earlier, COCHROMATIC NUMBER is NP-complete for permutation graphs and hence for circle graphs, whereas we have demonstrated that the problem is in P for chordal graphs (and hence for the properly contained family of interval graphs). One can think of circle graphs and circular arc graphs as the circular extensions of permutation graphs and interval graphs, respectively. Thus it would be of interest to know whether or not the complexity of COCHROMATIC NUMBER is the same for interval graphs and circular arc graphs. The status of this problem on circular arc graphs in unknown.

Acknowledgement

Part of this work was done when the last two authors were visiting at the University of Toronto. We thank NSERC for financial support, and Derek Corneil and the Department of Computer Science, University of Toronto, for invitations and hospitality. We also thank the referees for valuable comments and suggestions.

References