Common Connected Components of Interval Graphs

Michel Habib, Christophe Paul, Mathieu Raffinot

HAL Id: lirmm-00269438

https://hal-lirmm.ccsd.cnrs.fr/lirmm-00269438

Submitted on 3 Apr 2008

Common connected components of interval graphs

Michel Habib[*]   Christophe Paul[*]   Mathieu Raffinot[†]

Abstract

The Common Connected Problem (CCP) consists in identifying common connected components in two or more graphs on the same vertices (or reduced to). More formally, let $G_1(V, E_1)$ and $G_2(V, E_2)$ be two such graphs and let $V' \subset V$. If $G_1[V']$ and $G_2[V']$ are both connected, V' is said a common connected component. The CCP problem is the identification of maximal (for the inclusion order) such components, that form a partition of V. Let $n = |V|$ and $m = |E_1| + |E_2|$. We present an $O((n + m) \log n)$ worst case time algorithm solving the CCP problem when G_1 and G_2 are two interval graphs. The algorithm combines maximal clique path decompositions of the two input graphs together with an Hopcroft like partitioning approach.

1   Introduction

Let $G = (V, E)$ be a graph. The degree of a vertex $x \in V$ in the graph G is denoted by $d_G(x)$. If X is a subset of vertices of G, then we denote $G[X]$ the subgraph induced by X : the set of vertices of $G[X]$ is X and its edge set is $E_X = E \cap \{(u, v) \mid u \in X, v \in X\}$. We denote by $m_X = |E_X|$ the number of edges in $G[X]$ and by $|G[X]| = |X| + m_X$ the size of the induced subgraph. A connected component $X \subset V$ of G is such that $G[X]$ is connected. A connected component is maximal if it can not be augmented with other vertices.

In [9], the problem of finding common connected components of two graphs, namely the *CCP problem*, was addressed.

Definition 1 *A set $S \subseteq V$ of vertices is a* common connected component *of G_1 and G_2 if S is both a connected component in G_1 and G_2, and maximal for the inclusion order.*

As shown by the authors, it can easily be reduced to a pair of graphs G_1 and G_2 on the same vertex set V. We define $n = |V|$, $m_1 = |E_1|$, $m_2 = |E_2|$ (where E_1 and E_2 are respectively the edge sets of G_1 and G_2) and $m = m_1 + m_2$. It is easy to see that the set of common connected components of two graphs form a unique partition of V. The CCP problem, defined in [9], is that of identifying such a partition.

CCP Problem:

Input:	two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$.
Output:	the partition of V in common connected components.

---

[*]LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France, {habib,paul}@lirmm.fr

[†]Laboratoire Génome et Informatique, Tour Evry 2, 523, Place des Terrasses de l'Agora, 91034 Evry, France. raffinot@genopole.cnrs.fr
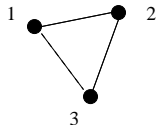
Figure 1: A simple counterexample showing that the set of common connected component of G_1 and G_2 is not the intersection of the connected components of the two interval graphs. The set $\{1, 2, 3\}$ is connected in G_1. The set $\{1, 3\}$ is connected in G_2 (in dashed edges). But $\{1, 3\}$ is not connected in $G_1[\{1, 3\}]$ and $G_2[\{1, 3\}]$.

A natural approach to solve CCP is to first search the maximal connected component of G_1. Then, in each of these components, search the connected component of G_2. In each such new connected component of G_2, search the maximal connected components of G_1, and repeat this process until the two sets of components on G_1 and G_2 are similar. It is not difficult to build examples where this approach leads to as many repetitions as the number of vertices. As each step is in $O(n + m)$ time, this approach is $O(n(n + m))$ worst case time. The algorithm proposed in [9] runs in $O(n \log n + m \log^2 n)$. Their algorithm mixes dynamical connectivity maintenance with a partitioning approach. Obtaining faster algorithms for solving CCP is a real challenge, since the graphs currently considered in computational biology are huge: studying graphs with more than $250\,000$ vertices becomes frequent (see for instance the TERAPROT project [21]). Reducing the complexity of the CCP problem remains an open problem even for some restricted graph families.

This paper considers the CCP problem on interval graphs. A graph is an *interval graph* iff there is a one-to-one mapping between its vertices and a set of interval on the real line such that two vertices are adjacent iff their corresponding intervals intersect. Considering interval graph make sense for both practical and theoretical reasons. First the CCP problem arises from applications in bioinformatics where graph comparisons is required [5]. Moreover many of the applications of the CCP problem in bioinformatics deal with interval graphs, that are the basic structure to represent chromosome built on smaller cDNAs. Consider for instance two interval graphs representing two different possible genomes, built on the same cDNA database. Comparing the longest "common" contigs, that is, the largest set of sequences that are linked together in the two genomes, require solving CCP. Interval graphs are also interesting for theoretical point of view since it is a highly structured graphs family.

In this paper we design an algorithm for CCP problem on interval graphs whose complexity is in $O((n + m) \log n)$ worst case time. The algorithm is both faster and simpler that the algorithm solving CCP on general graphs. It combines an Hopcroft like partitioning framework together with a kind of dynamical maintenance of a spanning separator forest. The intervals graphs are represented trough a forest of clique paths that roughly captures all the possible separators of the graphs. This forest is "dynamically" maintained, in the sense that we are able to fastly compute the new clique representation after extracting a set of vertices. Sets of vertices are extracted following an Hopcroft like partitioning framework, inspired by the gene teams identification algorithm [2] that has later been proved to resemble a simplified Hopcroft partitioning approach [5]. Notice that the gene team identification algorithm with $\delta = 1$, would solve CCP problem on proper interval graphs [2].

This article is organized as follows. We first explain in Section 2 the whole framework of a recursive partitioning algorithm to solve the CCP problem. Section 3 presents data-structures and algorithms that allows us to improve the time complexity for interval graphs. Finally the

whole algorithm and its complexity are explained and proved.

2   A recursive partitioning algorithm

Solving the CCP problem on two graphs G_1 and G_2 on the same vertex set V consists in computing a partition of V whose parts are the common connected component. A *partition* $\mathcal{P}$ of a set V is a set of disjoint subsets $\{\mathcal{X}_1, \ldots \mathcal{X}_k\}$, whose union is exactly V. Our partitioning algorithm is based on the following simple lemma.

Lemma 1 *Let G_1 and G_2 be graphs on the same vertex set V and let C be a maximal connected component of G_1 distinct from V. Then*

$$CCP(G_1, G_2) = CCP(G_1[C], G_2[C]) \cup CCP(G_1[V \backslash C], G_2[V \backslash C])$$

Proof. Let S be a common connected component. By definition S is connected in G_1. Since C is a maximal connected component, S is either included in C or in $V \setminus C$. It follows that any common connected component of G_1 and G_2 is either a common connected component of $G_1[C]$ and $G_2[C]$ or of $G_1[V \setminus C]$ and $G_2[V \setminus C]$. □

We can already derive from Lemma 1 a simple paradigm for a recursive algorithm. It takes as input two graphs G_1 and G_2 on the same vertex set V and a partition $\mathcal{P}$ of V. Initially, $\mathcal{P}$ is set to the trivial partition $\{V\}$. It first looks for a maximal connected component distinct from V of G_1 or G_2. If such a component C exists, according to Lemma 1 two recursive calls on the subgraphs induced respectively by C and $V \setminus C$ are launched.

CCP-Algorithm$(G_1 = (V, E_1), G_2 = (V, E_2))$
1.   **If** G_1 and G_2 are both connected **Then**
2.   **Return** $\mathcal{P} = \{V\}$
3.   **Else**
4.   **If** G_1 is not connected **Then**
5.   Let C be a connected component of G_1
6.   **Else**
7.   Let C be a connected component of G_2
8.   **End of if**
9.   Let $\mathcal{P}' =$**CCP-Algorithm**$(G_1[C], G_2[C])$
10.   Let $\mathcal{P}'' =$**CCP-Algorithm**$(G_1[V \setminus C], G_2[V \setminus C])$
11.   **Return** $\mathcal{P} = \mathcal{P}' \cup \mathcal{P}''$
12.   **End of if**

Figure 2: Recursive algorithm to compute the common connected components partition of the vertex set V of two graphs G_1 and G_2

Lemma 2 ***CCP-Algorithm*** *computes the common connected component partition of two graphs.*

Proof. First the algorithm ends since the recursive calls are done on strict subgraphs and it stops when both graphs are connected. The correctness of the algorithm directly follows from Lemma 1. □

Notice that the main difficulties of the **CCP-Algorithm** is first to compute a maximal connected component C of one the two input graph if one exists and then to extract the subgraphs induced by C and $V \setminus C$. Without any clever data-structure and advanced subroutine for these tasks, such a recursive approach yields to a $\mathcal{O}(n(n+m))$ worst case time.

Next section first presents algorithms for interval graphs that permit a $\mathcal{O}((n+m)\log n)$ worst case time. Section 3 is devoted to two algorithms that retrieve the maximal connected components of the subgraphs $G_1[C], G_2[C], G_1[V \setminus C], G_2[V \setminus C]$) after having extracted C. These algorithms strongly relies on interval graph structural properties. Their complexities are both $\mathcal{O}(|C| + m_C^d)$ where $m_C^d = \sum_{x \in C} d(x))$, which is proportional, not exactly to the size of their induced subgraph, but close to.

Choosing an arbitrary connected component for the recursive call is not enough to obtain the announced complexity, even the component can be extract in $\mathcal{O}(|C| + m_C^d)$. It would still lead to $\mathcal{O}(n(n+m))$ worst case time complexity. To lower the whole complexity, we combine the extraction scheme to an Hopcroft's partitioning approach. Only *small* maximal connected components have to be extracted. By small, we mean that the size of the connected component considered has to be less than or equal to the half of the size of the original graph. Such a component always exist if the graph is not connected. It would ensure that each vertex an edge is used at most $\log n$ time as expected for the announced complexity. This property is the basis of the Hopcroft's partitioning approach.

Generalization to an arbitrary number of graphs

One can also consider the k-CCP problem that is CCP problem applied to k graphs on the same vertex set. The result is a partition of the vertex set into maximal subset of vertices that induced connected subgraph of any into graph. It turns out that Lemma 1 can be generalized and the algorithm adpated.

Lemma 3 *Let $G_1, \ldots G_k$ be graphs on the same vertex set V and let C be a maximal connected component of G_1 distinct from V. Then*

$$k\text{-}CCP(G_1, \ldots G_k) = k\text{-}CCP(G_1[C], \ldots G_k[C]) \cup k\text{-}CCP(G_1[V \setminus C], \ldots G_k[V \setminus C])$$

Proof. Let S be a common connected component of the k graphs. By definition S is connected in G_1. Since C is a maximal connected component, S is either included in C or in $V \setminus C$. It follows that any common connected component of $G_1, \ldots G_k$ is either a common connected component of $G_1[C], \ldots G_k[C]$ or of $G_1[V \setminus C], \ldots G_k[V \setminus C]$. □

It is straightforward to modify the algorithm: the connected component C used has to be a connected component of an arbitrary graph among $G_1, \ldots G_k$. The complexity of the algorithm just increases by a factor k.

3 Clique path representation of interval graphs

This section present the material for the interval graphs. We first introduce some well-known properties and the data-structures used in the algorithms. Then two algorithms that updates the data-structures for induced subgraphs are depicted. These algorithms permit efficient recursive calls. Finally, the last algorithm looks for a small connected component of a given interval graph. It will be used at lines 4-8.

3.1 Preliminaries and data-structures

Let $G = (V, E)$ be a graph and $G[X]$ the subgraph induced by X. We set $m_X^d = \sum_{x \in X} d_G(x)$. A *clique* is a complete induced subgraph (not necessarily *maximal* for the inclusion).

A graph is an *interval graph* iff it is the intersection graph of a family of intervals on the real line: there is a one-to-one mapping between the intervals and the vertices of the graphs, and two vertices are adjacent iff their corresponding intervals intersect [18]. It follows that the family of interval graphs is *hereditary*: any induced subgraph of an interval graph is an interval graph.

Definition 2 *Let $G = (V, E)$ be a connected interval graph. A* clique decomposition path *of G is a path $P = (\mathcal{C}, F)$ such that:*

1. *any set $C \in \mathcal{C}$ is a set of vertices and $\bigcup_{C \in \mathcal{C}} C = V$;*

2. *any $(u, v) \in E$ is contained in some $C \in \mathcal{C}$;*

3. *the set $\mathcal{C}_u = \{C \in \mathcal{C} \mid u \in C\}$ induces a subpath P_u of P*

4. *any $C \in \mathcal{C}$ is a clique;*

A clique decomposition path will be denoted hereafter by CDP. Notice that a CDP gives an interval intersection model of the interval graph: the underlying path P and the family of subpaths P_u that contains the vertex u. If the condition 4 is not required, a decomposition path can be defined for arbitrary graphs and this is the basement of the pathwidth theory (see [3]).

Dealing with interval graphs, we usually define the *Maximal Clique decomposition Path* (shorten by MCP) where any clique $C \in \mathcal{C}$ has to be a maximal clique.

A *separator* is a set S of vertices whose removal disconnects the graph in several connected components. A separator S is *minimal* if there exists a pair of vertices u, v such that no subset of S separates u and v in different connected components. Since interval graphs are chordal (graphs with no induced cycle of length larger than 3), any minimal separator is a clique [8]. The following lemma gives some hints on the structure of the set of minimal separator of an interval graph.

Lemma 4 (eg. [10])

- *Let P be a MCP of an interval graph G. A set of vertices S is a minimal separator iff it is the intersection S of some consecutive cliques C_1 and C_2 in P.*

- *Let P be a MCP of an interval graph G. The intersection of any pair of consecutive cliques is a separator.*

For our needs, we label the edges of a CDP by the intersection of the corresponding cliques. A non-connected interval graph clearly enjoys a CDP: the edges between two cliques of different paths are labelled by the empty set since these cliques belong to different connected components and are disjoint. The number of cliques in a CDP P is denoted $|P|$. We say that the set of paths defines a *linear forest* denoted CPF for Clique Path Forest. When all the paths are maximal, the forest is denoted MCPF.
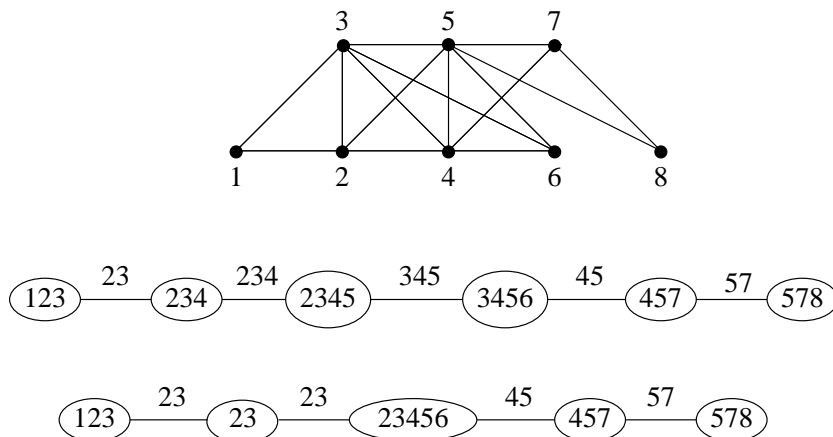
Figure 3: An interval graph with two CDPs. The second is maximal.

Lemma 5 (eg [10]) *Let G be an interval graph with n vertices and m edges. Any MCP is of size $\mathcal{O}(n+m)$.*

Many linear time interval graphs recognition algorithms exist. The first one is due to Booth and Lueker in 1976 [4]. Most recent ones [14, 7] are much simpler than the original. All these algorithms are able to output in $O(n+m)$ a maximal clique path decomposition.

For algorithmic settings, in case of non-connected interval graphs, the set of paths of a CPF is stored in a list $\mathcal{F}$. The cliques of a CDP are stored in a doubly linked list and the CDP are rooted on one of their extremities. A given clique C stores a pointer to its *father* $f(C)$ and to its *son* $s(C)$; its set of vertices is stored in a doubly linked list and its size is denoted n_C. Moreover, each edge is assigned to a record containing: (a) its two extremities; (b) the label of its minimal separator (see lemma 4) whose vertices are stored in a doubly linked list; (c) the size n_S of this separator. In addition, two lists, namely L_S and L_C, are associated to any vertex x. The list L_S (resp. L_C) contains pointers to the copies of x in each separator (resp. clique) containing x.

3.2 Dynamic clique decomposition path

Lemma 6 *Let $P = (\mathcal{C}, F)$ be a CDP of $G = (V, E)$ and $X \subseteq V$. Then $P' = (\mathcal{C}', F')$ defined as follows is a CDP of $G[V \backslash X]$.*

- *$\mathcal{C}' = \{f(C) \mid C \in \mathcal{C}\}$, where $f(C) = C \backslash X$;*

- *$(f(C_1), f(C_2)) \in F'$ iff $(C_1, C_2) \in F$;*

- *$(f(C_1), f(C_2))$ is labelled by $f(C_1) \cap f(C_2) = (C_1 \cap C_2) \backslash X$.*

Proof. Let us consider two vertices u and v belonging to $V \backslash X$. The proof follows from the definition. First any $f(C)$ is a clique and $\bigcup_{C \in \mathcal{C}} f(C) = V \backslash X$. If u and v are adjacent, there exists a clique $C \in \mathcal{C}$ containing both u and v. Clearly $f(C)$ also contains both u and v. Since the set $\mathcal{C}_u \subset \mathcal{C}$ of cliques containing $u \in V \backslash X$ occurs consecutively in P, the set $\mathcal{C}'_u \subset \mathcal{C}'$ also occurs consecutively in P'. $\square$

Notice that some separator may be empty after the extraction of X, in which case the resulting CDP is in fact a CPF.

For complexity issue, the above operation is implemented by two different algorithms. Given a CDP, the first one removes the vertices of a given set from each clique: it is called REMOVE. The second, in contrast, computes the intersection of any clique with a given set: it is called EXTRACT.

REMOVE(P, X)
1.   Let $\mathcal{F}$ be a linear forest containing P
2.   **For** any $x \in X$ **Do**
3.   **For** any clique C st $x \in C$ **Do**
4.   Remove x from C
5.   Decrease n_C by 1
6.   **End of for**
7.   **For** any separator S between cliques C and C' st $x \in S$ **Do**
8.   Remove x from S
9.   Decrease n_S by 1
10.   **If** $n_S = 0$ **Then**
11.   Let (P, C_r) be the CDP containing the edge (C, C') labeled by S
12.   Remove the edge (C, C') from P (C is the father of C')
13.   Create in $\mathcal{F}$ the new CDP (P', C')
14.   **Else**
15.   **If** $n_S = n_C$ **Then**
16.   Remove C from P
17.   Connect the $s(C)$ and $f(C)$ with the edge labelled by $s(C) \cap f(C)$
18.   **End of if**
19.   **If** $n_S = n_{C'}$ **Then**
20.   Remove C' from P
21.   Connect the $s(C')$ and $f(C')$ with the edge labelled by $s(C') \cap f(C')$
22.   **End of if**
23.   **End of if**
24.   **End of for**
25.   **End of for**
26.   **Return** $\mathcal{F}$

Figure 4: Maintaining a MCPF of a graph after removing a set of vertices X from an interval graph represented by a MCP.

The pseudo-code of algorithm REMOVE(P, X) is given in Fig. 4. The next lemma 7 states its validity and time complexity.

Lemma 7 *Let P be a MCP of the connected interval graph $G = (V, E)$. The algorithm* REMOVE(P, X) *(Fig.4) computes in $O(|X| + m_X^d)$ time a linear forest of $G[V \backslash X]$ where each path is a MCP of the corresponding connected component.*

Proof. First notice that by lemma 6, when the vertices of X have been removed (lines 4 and 8), $\mathcal{F}$ is a CDP of $G[V \backslash X]$ (but no longer a maximal one). Let S be the intersection between two consecutive cliques C and C' of a given path $P \in \mathcal{F}$ (w.l.o.g. $C = f(C')$).

- $S = \emptyset$ (S is no longer a separator since it is empty): Since P is a CDP C and C' belongs to different connected components and P can be split into two CDPs. The first one contains the clique from the root to C while the second one is rooted at C' and contains the clique descending from C' (lines 10-13).

- $n_S = n_C$ (the case $n_S = n_{C'}$ is similar): C is no longer a maximal clique (it is included in C'). Therefore we can remove C from P (lines 15-22).

It follows that when $\mathcal{F}$ has been cleaned up, any clique is a maximal clique and each new CDP is therefore a MCP. For complexity issue, since the number of copies of elements of X is $\mathcal{O}(|X| + m_X^d)$ and each copy is touched once, removing X cost $\mathcal{O}(|X| + m_X^d)$. The cleaning can be done within the same complexity since (a) removing a separator or a clique costs $\mathcal{O}(1)$; (b) the number of removing operations is bounded by the number of copies of elements of X. $\square$

We now consider the maximal clique path decomposition of the induced subgraph $G[X]$. The pseudo-code of algorithm $\texttt{EXTRACT}(P, X)$ is given in Fig. 5. The next lemma 8 states its validity and time complexity.

EXTRACT(P, X)
1.   Let $\mathcal{F}$ be an empty linear forest
2.   **For** any $x \in X$ **Do**
3.   **For** any clique C containing x **Do**
4.   **If** C has not been already duplicated **Then**
5.   Create a copy $C' = \{x\}$ in $\mathcal{F}$ in a new singleton CDP
6. $n_{C'} \leftarrow 1$
7.   **Else**
8.   Let C' the existing copy of C
9. $C' \leftarrow C' \cup \{x\}$
10. $n_{C'} \leftarrow n_{C'} + 1$
11.   **End of if**
12.   **End of for**
13.   **For** any separator S containing x **Do**
14.   Let (C_1, C_2) be the edge labeled by S in P (wlog $C_1 = f(C_2)$)
15.   **If** (C_1, C_2) has not been duplicated **Then**
16.   Create a new edge (C_1', C_2') labelled by x
17. $n_{S'} \leftarrow 1$
18.   **Else**
19.   Let S' the label of the edge (C_1', C_2')
20. $S' \leftarrow S' \cup \{x\}$
21. $n_{S'} \leftarrow n_{S'} + 1$
22.   **End of if**
23.   **End of for**
24.   **End of for**
25.   Remove from $\mathcal{F}$ any non maximal clique as in lines 15-22 of REMOVE (fig 4)
26.   **Return** $\mathcal{F}$

Figure 5: Maintaining a MCPF of the induced subgraph of $G[X]$ when extracting X from G.

Lemma 8 *Let F be a MCP of the connected interval graph $G = (V, E)$. The algorithm* **EXTRACT**(F, X) *(Fig. 5) computes in $\mathcal{O}(|X| + m_X^d)$ time a linear forest of $G[X]$ where any path is a MCP.*

Proof. A similar proof than that of lemma 7 shows that a linear forest of $G[X]$ can be computed in $\mathcal{O}(|X| + m_X^d)$. □

3.3 Smaller Induced Subgraph (SIS) algorithm

The SIS algorithm on two MCPs P_1 and P_2 allows us to find the smallest of the two induced subgraphs in time proportional to the size of this smallest subgraph. The difficulty comes from that the sizes of the two paths are not necessarily representative of the sizes of their induced subgraphs. It may happen that $|P_1| < |P_2|$, but that $|G[V_1]| > |G[V_2]|$, where V_1 (resp. V_2) is the set of vertices contained in the cliques of P_1 (resp. P_2).

To overcome this obstacle, we use a trick. We perform in parallel a Depth First Search (DFS) on the two paths. In "parallel" means that we read a new clique (or path node) of each MCP in alternance. During this search, we compute for each path the sums S_1 and S_2 of the sizes of the cliques we encountered.

At the end of this parallel DFS, the smallest MCP, say P_1, has been totally covered, and S_1 is the size of its induced subgraph. If $S_1 \leq S_2$, the simplest case (a), the subgraph induced by P_1 is smaller than that induced by P_2, and SIS returns P_1.

Otherwise, if $S_1 > S_2$ we continue the DFS of the second path P_2, computing the new sum S_2' for each new clique encountered. Figure 6 illustrates this search.
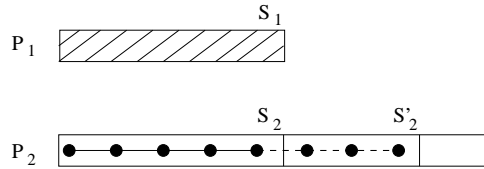

Figure 6: Continuing the Depth First Search in the longest path P_2 if $S_1 > S_2'$ while until either $S_2' \geq S_1$ or P_2 is completely covered.

The process goes on, until, case (b), either the whole path P_2 has been visited, in which case $S_2' \leq S_1$, either, case (c), S_2' becomes greater than S_1. Figure 7 illustrates these two last cases.

It is obvious that SIS returns the path which represents the smallest induced graph. We prove in the next lemma 9 that its complexity only depends of the size of this smallest induced subgraph.

Lemma 9 *Algorithm SIS applied on two paths P_1 and P_2 returns which of both represents the smallest induced subgraph $G[X]$ in $O(|X| + m_X)$.*

Proof. Let c_1 (resp. c_2) be the number of cliques visited in P_1 (resp c_2) at the end of SIS algorithm. The total number of cliques visited is $c_1 + c_2$.

In case (a), P_1 represents the smallest subgraph $G[X]$ of size $S_1 = |X| + m_X$. The number $c_1 + c_2$ is in this case $2c_1$. As $c_1 \leq S_1$ (lemma 5), the complexity of SIS is $O(|X| + m_X)$.

In case (b), $S_2' \leq S_1$. The path P_1 represents the smallest subgraph $G[X]$ of size $S_2' = |X| + m_X$. Therefore SIS returns P_2. As the first DFS stopped first w.l.o.g on P_1, $c_2 > c_1$, and, as $c_2 \leq S_2'$ (lemma 5), $c_1 + c_2 < 2S_2'$ and the complexity of SIS is $O(|X| + m_X)$.

In case (c), the path P_1 represents the smallest subgraph $G[X]$ of size $S_1 = |X| + m_X$. As $c_2 \leq S_1 + 1$, $c_1 + c_2 \leq 2S_1 + 1$ and the complexity of SIS is $O(|X| + m_X)$. □

(b) The longest path P_2 is totally covered by continuing the DFS.

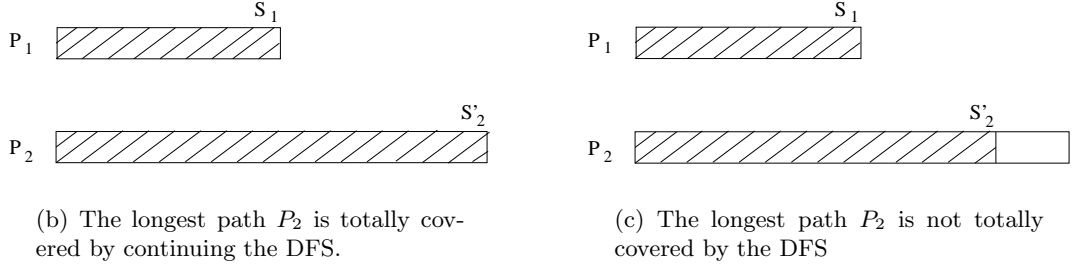(c) The longest path P_2 is not totally covered by the DFS

Figure 7: Two ending cases when continuing the DFS on the longest path P_2. In the first case (a), the DFS covers all the vertices of P_2. Then as $S_2' \leq S_1$, SIS returns P_2. In the second case, at most S_1 vertices of P_2 have been visited by the DFS without exploring all the tree. Then SIS returns P_1.

4   The whole CCP algorithm for interval graphs

The whole CCP algorithm for interval graphs (CCPI-Algorithm) is shown on figure 8. The algorithm takes as input two lists L_1 and L_2 are respectively the clique forest decompositions of the two graphs G_1 and G_2. It outputs a partition of the common vertex set. At lines 5 and 7, it looks for a connected component C whose size is at most half of the size of the corresponding graph. By lemma 9, it can be done in time proportional to the connected component. Let P be the CDP of C. W.l.o.g. we assume that C is a connected component of G_1 and P is the first CDP of L_1. Lines 11 and 12 compute the four subgraphs on which the recursive calls will be done. Using EXTRACT(L_2,P) and REMOVE(L_2,P) we compute the subgraphs of G_2 respectively induced by the vertices V_P belonging to the cliques of P and $V \setminus V_P$. As seen in Lemmas 8 and 7, it can be done in $\mathcal{O}(|V_P| + m_{V_P}^d)$.

Theorem 1 *The CCPI-algorithm applied on $MCPF(G_1)$ and $MCPF(G_2)$ correctly identifies the common connected components of G_1 and G_2.*

Proof. The CCPI-Algorithm fully respects the general algorithm framework described in section 2. Indeed lemmas 8 and 7 ensures that the recursive calls are done on the right subgraphs. The only difference is that we now choose which maximal component we extract first. □

To analyse its complexity, we use an amortized argument that is common to many Hopcroft like approach, but did not appear in the original paper [17]. To our knowledge, it is due to [6].

Theorem 2 *The CCPI-Algorithm is worst case $\mathcal{O}((n + m) \log n)$ time .*

```
CCPI-Algorithm(L_1, L_2)
  1.      If |L_1| = 1 and |L_2| = 1 Then
  2.          Return P = {V}              /*G_1 and G_2 are both connected*/
  3.      Else
  4.          If |L_1| ≥ 2 Then
  5.              P ← SIS(L_1[1], L_1[2])     /*G_1 is not connected*/
  6.          Else
  7.              P ← SIS(L_2[1], L_2[2])     /*G_2 is not connected*/
  8.          End of if
  9.                  /* we assume below w.l.o.g that P = L_1[1] */
 10.          L'_1 ← P;                    L''_1 ← L_1\P
 11.          L'_2 ← EXTRACT(L_2, V[P]);    L''_2 ← REMOVE(L_2, V[P])
 12.          Let P' =CCP-Algorithm(L'_1, L'_2)
 13.          Let P'' =CCP-Algorithm(L''_1, L''_2)
 14.          Return P = P' ∪ P''
 15.      End of if
```

Figure 8: Recursive algorithm to compute the partition of the vertex set into common connected components of two interval graphs G_1 and G_2 represented respectively by the MCPF $L1$ and L_2.

Proof. We first focus on the number of times a vertex x and a transition (y, z) may participate to EXTRACT and REMOVE. W.l.o.g., let S_1 be the size of the subgraph of G_1 at the beginning of a recursive call of CCCIA. If a maximal connected component of G_1 is extracted trough its MCP P, then the size of the induced subgraph of P is less than or equal to $S_1/2$. This is straightforward since P has been isolated trough SIS as the smallest of the two induced subgraphs. By induction, if x and (y, z) participate to many EXTRACT and REMOVE, they are contained in subgraphs whose sizes is divided at least by two at each recursive call. Therefore, they may only participate to $\log(n+m)$ EXTRACT and REMOVE calls.

Secondly, we amortized the cost of each EXTRACT and REMOVE of a path P on all the vertices and edges of the induced subgraph of P. The complexity of EXTRACT and REMOVE (lemmas 8 and 7)) for extracting a set X out of a graph G is $|X| + m_G^d(X)$. We amortize the cost $|X|$ over each vertex of X, and therefore a vertex $x \in X$ participates for a constant amount of time. The term $m_G^d(X)$ is amortized over the edges. As an edge (x, y) may be visited when considering x and when considering y, an edge can be visited only twice and consecutively participates for a constant amount of time.

In consequence, each vertex or each edge cost at most $\log(n+m)$. This leads to an overall complexity of $\mathcal{O}((n + m) \log(n + m))$. As in the worst case, $m = \mathcal{O}(n^2)$, the final complexity is $\mathcal{O}((n + m) \log n)$ worst case time. □

The space complexity is $O((n + m))$, since the two MCPFs are space linear in $n + m$ and that the recursive call of CCCIA algorithm can be managed with a list of at least $O(\log(n+m))$ pointers on the MCPFs.

5 Conclusion

We presented an $\mathcal{O}((n+m)\log n)$ worst case time and space complexity for solving the CCP problem on interval graphs. Let us recall that in the general case, the best algorithm runs in $\mathcal{O}(n\log n + m\log^2 n)$ [9]. The algorithm combines an Hopcroft partitioning approach with a maintenance of a spanning clique forest decomposition of the two graphs. Designing faster algorithms or proving a lower bound for CCP remains open, on interval and general graphs. It is worthwhile to notice that even on chordal graph the general upper bound can still not be improved.

References

[1] D. Beauquier and J. Berstel an P. Chrétienne, editors. *Eléments d'algorithmique*. Masson, Paris, 1992.

[2] A. Bergeron, S. Corteel, and M. Raffinot. The algorithmic of gene teams. In *Workshop on Algorithms in Bioinformatics (WABI)*, number 2452 in Lecture Notes in Computer Science, pages 464–476. Springer-Verlag, Berlin, 2002.

[3] H. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1-2), 1993.

[4] K.S. Booth and G.S. Lueker. Testing for the consecutive ones properties, interval graphs and graph planarity using pq-tree algorithm. *J. Comput. Syst. Sci.*, 13:335–379, 1976.

[5] M.-P. Béal, A. Bergeron, and M. Raffinot. Gene Teams and Hopcroft's Partionning Framework. 2003. Submitted.

[6] A. Cardon and M. Crochemore. Partitioning a graph in $O(|A|\log_2 |V|)$. *Theoretical Computer Science*, 19(1):85–98, 1982.

[7] D.G. Corneil, S. Olariu, and L. Stewart. The ultimate interval graph recognition algorithm? In *Proceedings of the ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 175–180, 1998.

[8] G.A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Uni. Hamburg*, 25, 1961.

[9] A.-T. Gai, M. Habib, C. Paul, and M. Raffinot. Identifying Common Connected Components of Graphs. *Technical report LIRMM-03016*, 2003. Submitted.

[10] P. Galinier, M. Habib, and C. Paul. Chordal graphs and their clique graph. In M. Nagl (Ed.), editor, *Graph-Theoretic Concepts in Computer Science, WG'95*, volume 1017 of *Lecture Notes in Computer Science*, pages 358–371, Aachen, Germany, June 1995. 21st Internationnal Workshop WG'95, Springer.

[11] P. Galinier, M. Habib, and C. Paul. Chordal graphs and their clique graphs. In *Workshop on Graph-Theoretic Concepts in Computer Science*, pages 358–371, 1995.

[12] F. Gavril. The intersection graphs of a path in a tree are exactly the chordal graphs. *Journ. Comb. Theory*, 16:47–56, 1974.

[13] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, 1980.

[14] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234:59–84, 2000.

[15] M. Habib, C. Paul, and L. Viennot. A synthesis on partition refinement: a useful routine for strings, graphs, boolean matrices and automata. In *15th Symposium on Theoretical Aspect of Computer Science (STACS)*, number 1373 in Lecture Notes in Computer Science, pages 25–38. Springer-Verlag, Berlin, 1998.

[16] M. Habib, C. Paul, and L. Viennot. Partition refinement techniques: an interesting algorithmic tool kit. *International Journal of Foundations of Computer Science*, 10(2):147–170, 1999.

[17] J. E. Hopcroft. An $n \log n$ algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *The Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

[18] C.G. Lekkerkerker and J.C. Boland. Representation of a finite graph by a set of intervals on the real line. *Fund. Math.*, 51:45–64, 1962.

[19] N. Luc, J.-L. Risler, A. Bergeron, and M. Raffinot. Gene Teams: A New Formalization of Gene Clusters For Comparative Genomics. *Computational Biology and Chemistry (ex. Computer and Chemistry)*, 2002. To appear.

[20] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[21] TERAPROT project. http://www.infobiogen.fr/services/teraprot/.