A Fully Dynamic Algorithm for Modular Decomposition and Recognition of Cographs

Ron Shamir

School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel

Roded Sharan

International Computer Science Institute, 1947 Center St., Suite 600, Berkeley CA 94704

Abstract

The problem of dynamically recognizing a graph property calls for efficiently deciding if an input graph satisfies the property under repeated modifications to its set of vertices and edges. The input to the problem consists of a series of modifications to be performed on the graph. The objective is to maintain a representation of the graph as long as the property holds, and to detect when it ceases to hold.

In this paper we solve the dynamic recognition problem for the class of cographs and some of its subclasses. Our approach is based on maintaining the modular decomposition tree of the dynamic graph, and using this tree for the recognition. We give the first fully dynamic algorithm for maintaining the modular decomposition tree of a cograph. We thereby obtain fully dynamic algorithms for the recognition of cographs, threshold graphs, and trivially perfect graphs. All these algorithms work in constant time per edge modification and $O(d)$ time per d-degree vertex modification.

Key words: Fully dynamic algorithm, cograph, recognition, modular decomposition.

1 Introduction

In a *dynamic graph problem* one has to maintain a graph representation throughout a series of on-line modifications, i.e., insertions or deletions of a

Email addresses: `rshamir@tau.ac.il` (Ron Shamir), `roded@icsi.berkeley.edu` (Roded Sharan).

vertex or an edge. The representation should allow to answer queries regarding certain properties of the dynamic graph, e.g., "is it connected?". Algorithms for the problem are called *dynamic* algorithms, and are categorized depending on the modification operations they support: An *incremental* (*decremental*) algorithm supports only vertex insertions (deletions). An *additions-only* (*deletions-only*) algorithm supports only edge additions (deletions). An *edges-only fully dynamic* algorithm supports both edge additions and edge deletions. A *fully dynamic* algorithm supports edge modifications as well as vertex modifications.

This paper investigates *dynamic recognition problems* in which the queries are of the form: "Does the graph belong to a certain class Π?". An algorithm for the problem is required to maintain a representation of the dynamic graph as long as it belongs to Π, and to detect when it ceases to belong to Π.

Several authors have studied the problem of dynamically recognizing specific graph families. Hell, Shamir and Sharan [1] have given a near optimal fully dynamic algorithm for recognizing proper interval graphs, which works in $O(d + \log n)$ time per modification involving d edges, i.e., $d = 1$ in case of an edge modification, and d is the degree in case of a vertex modification. (Throughout, we denote the number of vertices and edges in a graph by n and m, respectively.) Ibarra [2] has given an edges-only fully dynamic algorithm for chordal graph recognition, which handles each edge operation in $O(n)$ time, and an edges-only fully dynamic algorithm for split graph recognition, which handles each edge operation in constant time. Recently, Ibarra [3] has also devised an edges-only fully dynamic algorithm for interval graph recognition, which handles each edge operation in $O(n \log n)$ time. Incremental recognition algorithms were given by Hsu for interval graphs [4], and by Deng, Hell and Huang for connected proper interval graphs [5].

A very useful representation of a graph is its modular decomposition tree (we defer technical definitions to Section 2). The problem of generating the modular decomposition tree of a graph was studied by many authors and several linear-time algorithms were developed for it [6–8]. For the problem of dynamically maintaining the modular decomposition tree of a graph only two partial results are known. Muller and Spinrad [9] have given an incremental algorithm for modular decomposition, which handles each vertex insertion in $O(n)$ time. Corneil, Perl and Stewart [10] have given an optimal incremental algorithm for the recognition and modular decomposition of cographs, which handles the insertion of a vertex of degree d in $O(d)$ time.

In this paper we give the first fully dynamic algorithm for maintaining the modular decomposition tree of a cograph. Our algorithm works in $O(d)$ time per operation involving d edges. Based on this algorithm we develop fully dynamic algorithms for the recognition of cographs, threshold graphs and trivially perfect graphs. All these algorithms handle a modification involving d edges in $O(d)$ time. This is optimal with respect to all operations, with the possible exception of vertex deletion.

The paper is organized as follows: Section 2 contains the definitions and the terminology used in the paper. Section 3 presents the fully dynamic algorithm for recognizing cographs and maintaining their modular decomposition tree. Section 4 contains the recognition algorithms for threshold graphs and trivially perfect graphs.

2 Preliminaries

We provide here some basic definitions and background. We refer the reader to [11] for further background reading. All graphs in this paper are simple and undirected. Let $G = (V, E)$ be a graph. We denote its set of edges E also by $E(G)$. For a subset $R \subseteq V$ we denote by $G(R)$ the subgraph induced by the vertices in R. The *complement* of G is the graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(u, v) \notin E : u \neq v\}$. The *complement-connected components* of G are the connected components of $\overline{G}$. The graph P_4 is a path on four vertices. The graph C_4 is a cycle on four vertices. For a vertex $v \in V$ we denote by $N(v)$ the *open neighborhood* of v, consisting of all neighbors of v. We let $N[v] = N(v) \cup \{v\}$. For a new vertex $z \notin V$ and a set of edges E_z between z and vertices of V, we denote by $G \cup z$ the graph $(V \cup \{z\}, E \cup E_z)$ obtained by adding z to G. For a vertex $z \in V$ we denote by $G \setminus z$ the graph $G(V \setminus \{z\})$ obtained by removing z from G.

A *module* M in G is a set of vertices $M \subseteq V$ such that every vertex in $V \setminus M$ is either adjacent to every vertex in M, or non-adjacent to every vertex in M. A module M is called *trivial* if $M = V$ or M contains a single vertex. M is called *connected* if $G(M)$ is a connected subgraph. M is called *complement-connected* if $\overline{G(M)}$ is a connected graph. For brevity, we shall often refer to a module as if it was the subgraph induced by its vertices. (For example, we shall talk about the connected components of a module.) A disconnected module is called *parallel*. A complement-disconnected module is called *series*. A module which is both connected and complement-connected is called a *neighborhood* module. Note, that every module is exactly one of the three types: Series, parallel or neighborhood.

A module M is *strong* if for any module N with $N \cap M \neq \emptyset$, we have $N \subseteq M$ or $M \subseteq N$. A strong module M is a *maximal submodule* of a module $N \supset M$, if no strong submodule of N properly contains M and is properly contained in N. It has been shown (cf. [12]) that every vertex of a non-trivial module M is in a unique maximal submodule of M. Clearly, the maximal submodules of a parallel module are its connected components, and the maximal submodules of a series module are its complement-connected components. Hence, the structure of the modules of a graph G can be captured by the following *modular decomposition tree* T_G: The nodes of T_G correspond to strong modules of G. The root node is V, and the set of leaves of T_G consists of all the vertices of G. The children of every internal node M of T_G are the maximal submodules of M. Each internal node in T_G is labeled 'series', 'parallel', or 'neighborhood', depending on the type of its corresponding module. Note, that the modular decomposition tree of a given graph is unique.

In the sequel we denote the modular decomposition tree of a graph G by T_G. We refer to a node M of T_G by the set of vertices it represents, that is, the set of vertices in the leaves of the subtree rooted at M. For two vertices $u, v \in V$, we denote by M_{uv} the least common ancestor of $\{u\}$ and $\{v\}$ in T_G.

Let Π be a graph class. A *fully dynamic algorithm* for Π-recognition maintains a data structure of the current graph $G = (V, E)$ and supports the following operations:

- **Edge Insertion**: Given a non-edge $(u, v) \notin E$, update the data structure if $G \cup \{(u, v)\} \in \Pi$, or output *False* and halt otherwise.
- **Edge Deletion**: Given an edge $(u, v) \in E$, update the data structure if $G \setminus \{(u, v)\} \in \Pi$, or output *False* and halt otherwise.
- **Vertex Insertion**: Given a new vertex $v \notin V$ and a set of edges between v and vertices of G, update the data structure if $G \cup v \in \Pi$, or output *False* and halt otherwise.
- **Vertex Deletion**: Given a vertex $v \in V$, update the data structure if $G \setminus v \in \Pi$, or output *False* and halt otherwise.

Traditionally, fully dynamic algorithms handle only edge modifications, since vertex modifications can be performed by a series of edge modifications. (For example, in dynamic graph connectivity adding a vertex of degree d is equivalent to adding an isolated vertex, and then adding its edges one by one.) However, in our context we have to be more careful, since we may not be able to add or delete one edge at a time without ceasing to satisfy property Π (and even if there is a way to do that, it might be non-trivial to find it). In other words, adding or deleting a vertex can preserve the property, but adding or removing one edge at a time might fail to do so. Hence, vertex modifications must be handled separately by the dynamic algorithm.

2.1 A Reduction

A graph class Π is called *complement-invariant* if $G \in \Pi$ implies $\overline{G} \in \Pi$. Examples for complement-invariant classes include perfect graphs, cographs, split graphs, threshold graphs and permutation graphs.

We say that a dynamic algorithm *Alg* for recognizing some graph property is *based on modular decomposition* if: (1) *Alg* maintains the modular decomposition tree of the dynamic graph; and (2) the only operations that *Alg* makes are updates to the tree, or queries regarding the tree.

Observation 1 *The modular decomposition trees of a graph and its complement are identical up to exchanging the labels 'series' and 'parallel'.*

Theorem 2 *Let Π be a complement-invariant graph property. Let Alg be a dynamic algorithm for Π-recognition, which supports either edge insertions only or edge deletions only, and is based on modular decomposition. Then Alg can be extended to support both operations with the same time complexity.*

PROOF. Suppose that *Alg* is an additions-only algorithm. The proof for the case that *Alg* is a deletions-only algorithm is analogous. Let $G = (V, E)$ be the current graph. In order to delete an edge $(u, v) \in E$ we perform an insert operation on $\overline{G}$, by treating each parallel node in T_G as a series node and vice-versa. By Observation 1, the modular decomposition tree of $\overline{G}$ is identical to T_G up to exchanging the labels 'series' and 'parallel'. Since $\overline{\overline{G} \cup \{(u, v)\}} = G \setminus \{(u, v)\}$, the algorithm performs the update successfully if and only if $G \setminus \{(u, v)\} \in \Pi$. $\square$

3 Cographs

A graph is called a *cograph* (*complement reducible graph*) if it contains no induced P_4 [13]. This class of graphs is clearly complement-invariant. In this section we give a fully dynamic algorithm for recognizing cographs and maintaining their modular decomposition tree. The algorithm works in $O(d)$ time per operation involving d edges. It is based on the following fundamental characterization of cographs:

Theorem 3 ([13]) *A graph is a cograph if and only if its modular decomposition tree contains only parallel and series nodes.*

Another viewpoint on the modular decomposition tree of a cograph is as a method to build the graph: Going recursively up the tree, the subgraph of a parallel node is formed by taking the union of its children's subgraphs. For a series node, all edges between vertices in distinct child modules are added to that graph.

Theorem 3 implies that a cograph is either connected, or complement-connected, but not both. It also implies that in a modular decomposition tree of a cograph parallel and series nodes alternate along any path starting from the root. We use these facts often in the sequel.

3.1 The Data Structure

Let $G = (V, E)$ be the input graph. We maintain the modular decomposition tree T_G of G as follows: For each vertex of G we keep a pointer to its corresponding leaf-node in T_G. For each node M of T_G we keep its type, which can be 'series', 'parallel' or 'leaf', and its number of children. We also keep pointers from M to its parent and to its children. The parent pointer of the root node points to itself. In detail, each node M has an associated doubly linked list L. Each element of L corresponds to a child N of M, and consists of two pointers, one pointing to N and the other to M. The parent pointer of N points to its corresponding element in L. This data structure allows detaching a child from its parent in constant time. Note, that a node in T_G has no explicit record of the vertices that it contains as a module.

Initially T_G is calculated in linear time, e.g., using the algorithm of [10]. If G is discovered to contain an induced P_4 then our algorithm outputs *False* and halts. In the description below we assume that G is a cograph.

3.2 Adding an Edge

Let (u, v) be the edge to be added, and let $G' = G \cup \{(u, v)\}$. We observe that M_{uv} cannot be a series module since this would imply that the edge (u, v) is already present in G. Hence, by Theorem 3 M_{uv} is a parallel module. Let C_u and C_v denote the maximal submodules (equivalently, connected components) of M_{uv} which contain u and v, respectively. Without loss of generality, $|C_u| \leq |C_v|$. The edge insertion algorithm is based on the following theorem:

Theorem 4 *G' is a cograph if and only if $|C_u| = 1$ and v is adjacent to every other vertex in C_v.*

PROOF.

⇒ Suppose that $|C_u| > 1$. Then C_u contains some vertex a which is adjacent to u, and C_v contains some vertex b which is adjacent to v. Hence, $\{a, u, v, b\}$ induce a P_4 in G', so G' is not a cograph.

Suppose that $w \in C_v \setminus \{v\}$ is not adjacent to v. Let $v, x_1, \ldots, x_k = w$ be a shortest path from v to w in C_v, $k \geq 2$. Then $\{u, v, x_1, x_2\}$ induce a P_4 in G', so G' is not a cograph.

$\Leftarrow$ Suppose that G' contains an induced P_4. Since G is a cograph, an induced P_4 in G' must contain the edge (u, v). Suppose that $\{u, v, x, y\}$ induce a P_4 in G' (not necessarily in this order). One of x and y is therefore adjacent to exactly one of u and v. Without loss of generality, let x be adjacent to exactly one of u and v. Since every vertex in $V \setminus M_{uv}$ is either adjacent to both u and v, or non-adjacent to both of them, we have $x \in M_{uv}$. If $x \in C_u$, $|C_u| > 1$ and we are done. If $x \in C_v$, then x is adjacent to v and not to u. As $\{u, v, x, y\}$ induce a P_4, y is adjacent either to u only (out of u, v and x), or to x only. In the first case we have $y \in C_u$, implying that $|C_u| > 1$. In the latter case, we conclude that $y \in C_v$. But $(v, y) \notin E(G')$.

□

Note that the theorem implies that $\{v\}$ is a child of C_v in T_G, since otherwise the path from C_v to $\{v\}$ in T_G would contain a parallel node, and v would not be adjacent to all vertices of C_v.

Let us assume for now that G' is a cograph and we have already identified M_{uv}, C_u and C_v. We show below how to update T_G in this case. Later, we shall show how to check the conditions of Theorem 4 and how to find each of M_{uv}, C_u and C_v.

Let r be the number of children of M_{uv} in T_G. If both C_u and C_v contain a single vertex, we update T_G as follows: If $r = 2$, then the updates depend on the position of M_{uv} in T_G. If M_{uv} lies at the root of T_G, we change its label to 'series'. Otherwise, we connect $\{u\}$ and $\{v\}$ as children of the parent P of M_{uv} (which is a series module), and delete M_{uv}. If $r > 2$, we make $\{u\}$ and $\{v\}$ the children of a new series node $\{u, v\}$, and connect this node as a child of M_{uv}.

Suppose now that $|C_v| > 1$. By Theorem 4 (since G' is a cograph) $|C_u| = 1$ and v is adjacent to every vertex in $C_v \setminus \{v\}$. We update T_G by first detaching $\{u\}, \{v\}$ and C_v from their parents and forming a new parallel node $K = \{u\} \cup (C_v \setminus \{v\})$. We continue according to one of the following cases:

(1) $r > 2$: We add a new series node $\{u\} \cup C_v$ as a child of M_{uv}. We then make $\{v\}$ and K the children of $\{u\} \cup C_v$.

(2) $r = 2$: We connect $\{v\}$ and K to the parent node of M_{uv} (which might be M_{uv} itself if it is the root). We then delete M_{uv}, unless it lies at the root of T_G, in which case we change its label to 'series'.

It remains to describe the subtree of $T_{G'}$ rooted at the new parallel node K. Let $K_1, \ldots, K_l, \{v\}$ be the complement-connected components of C_v. There are two cases to consider:

(1) $l > 1$: In this case $C_v \setminus \{v\}$ is necessarily connected. Hence, we need to make $\{u\}$ and $C_v \setminus \{v\}$ the children of K, and connect $K_1, \ldots, K_l$ to $C_v \setminus \{v\}$ as its children (see Figure 1). In order to carry out these changes efficiently, we do not introduce a new node $C_v \setminus \{v\}$. Instead, we make C_v the child of K. Since a node has no record of its corresponding vertex set, this alternative update is equivalent to the requested one. Correspondingly, we shall now refer to the former node C_v as $C_v \setminus \{v\}$.

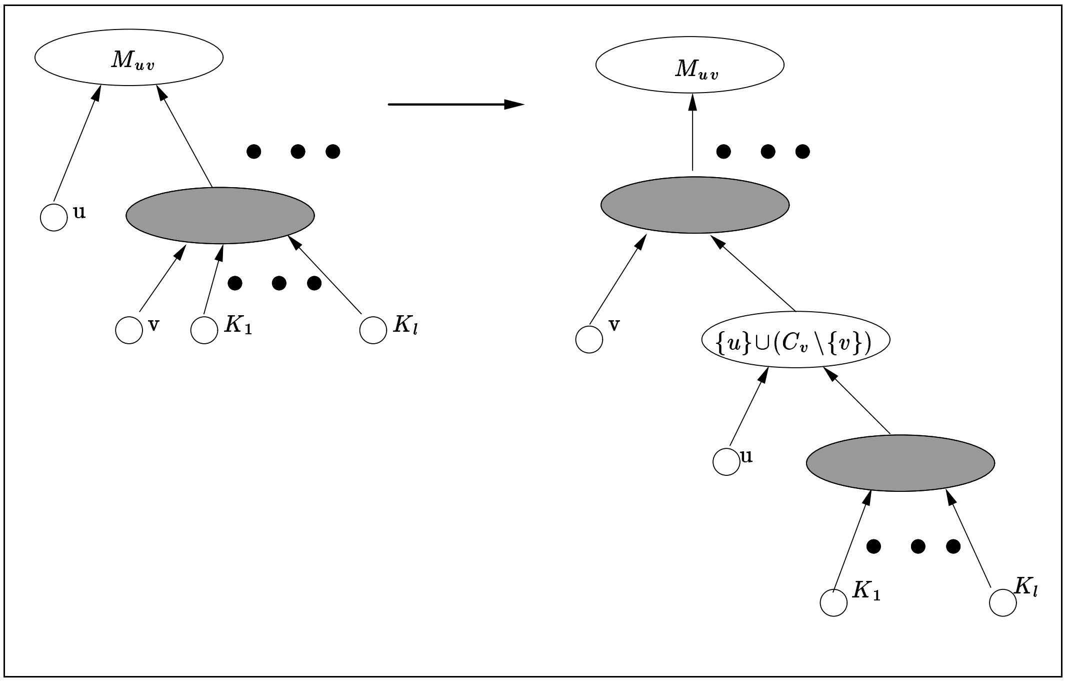

Fig. 1. The updates to the modular decomposition tree in case M_{uv} and C_v have more than two children each, and $|C_u| = 1$. Series nodes are drawn shaded.

(2) $l = 1$: If $K_1 = C_v \setminus \{v\}$ contains a single vertex w, we make $\{u\}$ and $\{w\}$ the children of K. Otherwise, K_1 is complement-connected and, therefore, it is disconnected. Let $J_1, \ldots, J_p$ be the connected components of K_1, $p \geq 2$. Then we need to make $\{u\}$ and $J_1, \ldots, J_p$ the children of K. Instead of introducing the new node K, we make (the former node) K_1 a child of $\{u\} \cup C_v$ (in addition to $\{v\}$), and attach $\{u\}$ as an additional child of K_1. Finally, we delete C_v.

Obviously, all the above updates to T_G can be carried out in constant time. Updating the number of children at each node can be also supported in constant time. It remains to show how to find M_{uv}, C_u and C_v efficiently, and how to verify the conditions of Theorem 4. In other words, we have to check if one of $\{u\}$ and $\{v\}$ is a child of M_{uv}, and the other is connected to every vertex in its connected component in $G(M_{uv})$. It is straightforward to see that this is the case if and only if M_{uv} is parallel and is either the parent of $\{u\}$ and the grandparent of $\{v\}$, or vice versa (assuming that $|C_u| > 1$ or $|C_v| > 1$). One can determine if such a configuration exists in constant time, by checking if the parent of $\{u\}$ ($\{v\}$) is parallel, and coincides with the grandparent of $\{v\}$ ($\{u\}$). If such a configuration exists, then it immediately identifies M_{uv}, C_u and C_v, and we update T_G accordingly. Otherwise, the algorithm outputs *False* and halts.

The following theorem and corollary summarize our results:

Theorem 5 *There is an optimal additions-only algorithm for recognizing cographs and maintaining their modular decomposition tree, which handles each edge insertion in constant time.*

Corollary 6 *There is an optimal edges-only fully dynamic algorithm for recognizing cographs and maintaining their modular decomposition tree, which handles each edge modification in constant time.*

3.3 Vertex Modifications

We shall generalize our algorithm to handle vertex insertions and deletions as well. Supporting vertex insertions is based on the incremental algorithm for cograph recognition of Corneil et al. [10]. This algorithm handles the insertion of a vertex of degree d in $O(d)$ time, updating the modular decomposition tree accordingly, and can be supported by our data structure with some trivial extensions.

It remains to show how to handle the deletion of a vertex u of degree d from G. Let $G' = G \setminus u$. G' is a cograph as an induced subgraph of G. Hence, we concentrate on updating T_G. Let P be the parent node of $\{u\}$ in T_G. There are four cases to consider:

(1) If T_G contains $\{u\}$ only, then $T_{G'}$ is empty.
(2) If P has at least three children then $T_{G'}$ is obtained from T_G by deleting $\{u\}$.
(3) If P has only two children that are both leaves, $\{u\}$ and $\{v\}$, then $T_{G'}$ is obtained from T_G by deleting $\{u\}$ and replacing P with $\{v\}$.
(4) If P has only two children $\{u\}$ and M, where M is an internal node of T_G, then two cases are possible:
    (a) If P lies at the root of T_G, then $T_{G'}$ is the subtree of T_G which is rooted at M.
    (b) Otherwise, let F be the parent of P. Then $T_{G'}$ is formed from T_G by connecting the children of M to F, and deleting $\{u\}, P$ and M.

Proposition 7 *The deletion of a vertex u of degree d can be handled in $O(d)$ time.*

PROOF. All cases except 4b can be handled in constant time. Consider this last case. If P is a series module, then u is adjacent to all vertices of M, and $T_{G'}$ can be constructed in $O(d)$ time. If P is a parallel module, then instead of deleting M we replace F with M, attaching the former children of F (except P) as children of M. Since u is adjacent to all the vertices of these children modules, this takes $O(d)$ time. □

We are now ready to state our main result:

Theorem 8 *There is a fully dynamic algorithm for recognizing cographs and maintaining their modular decomposition tree, which handles insertions and deletions of vertices and edges, and works in $O(d)$ time per operation involving d edges.*

4 Subclasses of Cographs

4.1 Threshold Graphs

A graph $G = (V, E)$ is called a *threshold graph* if there exist non-negative real numbers w_v, $v \in V$ and t such that for every $U \subseteq V$, $\sum_{v \in U} w_v \leq t$ if and only if U is an independent set [14]. We use the following characterization of threshold graphs:

Theorem 9 (cf. [11]) *A graph is a threshold graph if and only if it is both a cograph and a split graph.*

We also use the split recognition algorithm of Ibarra [2], which handles insertions and deletions of edges in constant time. Ibarra's algorithm builds on a characterization of split graphs by their degree sequence [15]. Upon each modification it updates the degree sequence of the dynamic graph and checks if it continues to satisfy the split graph characterization.

Theorem 10 *There is a fully dynamic algorithm for threshold recognition, which works in $O(d)$ time per operation involving d edges.*

PROOF. By Theorem 8 there exists a fully dynamic algorithm A_1 for cograph recognition with the same time bounds. By a simple generalization of the split recognition algorithm of Ibarra [2], one can obtain a fully dynamic algorithm A_2 for split recognition, which handles also vertex modifications, and works in $O(d)$ time per modification involving d edges. Our algorithm for threshold recognition executes A_1 and A_2 in parallel, and upon a modification outputs *False* and halts if and only if any of these algorithms outputs *False*. □

4.2 Trivially Perfect Graphs

A graph is called *trivially perfect* if it is a cograph and contains no induced C_4 [16]. Note that this class of graphs is not complement-invariant. For example, the graph C_4 is not trivially perfect, but its complement (a pair of independent edges) is. In this section we present a fully dynamic algorithm for trivially perfect graph recognition. Our algorithm is an extension of the cograph recognition algorithm, which after each modification checks also whether the current graph contains an induced C_4.

Suppose that $G = (V, E)$ is a trivially perfect graph. If we delete a vertex from G then the resulting graph is clearly trivially perfect. If we add an edge to G and the new graph is a cograph, then it is also a trivially perfect graph. This follows by noting that if an induced C_4 is created, then G must have contained an induced P_4. Hence, it suffices to show how to check for the existence of an induced C_4 after edge deletions and vertex insertions. We assume in the following that the current graph G is trivially perfect, and that the modified graph G' is a cograph, as otherwise, the cograph recognition algorithm outputs *False* and we are done.

4.2.1 Adding a Vertex

Let z be a new vertex of degree d to be added, and let $G' = G \cup z$. Clearly, if G' contains an induced C_4, it is of the form $\{a, b, c, z\}$ for some vertices $a, b, c \in V$ (where (a, c) and (b, z) are the non-edges).

If z connects two or more connected components of G then it must be adjacent to every vertex in these components, or else G' would contain an induced P_4. Therefore, in this case G' is a trivially perfect graph. If z is adjacent to all vertices of a single component then again G' is trivially perfect. One of these cases applies if and only if $\{z\}$ is either a child of a series root module (if G' contains a single connected component), or a grandchild of a parallel root module (if G' contains more than one component). We can check for such configurations in constant time. The remaining case is when z is adjacent to some but not all vertices of a single connected component C of G. We handle this case below.

Lemma 11 *A cograph contains an induced C_4 if and only if its modular decomposition tree has a series node with at least two non-trivial children.*

PROOF. If H is a cograph and $\{a, b, c, d\}$ induce a C_4 in H, then the least common ancestor of $\{a\}$,$\{b\}$,$\{c\}$ and $\{d\}$ in T_H is a series module with at least two non-trivial maximal submodules (one containing a, c and the other containing b, d).

Conversely, if the modular decomposition tree of a cograph H contains a series node with two non-trivial children M_1 and M_2, then any two vertices from M_1 together with any two vertices from M_2 induce a C_4 in H. $\square$

Lemma 11 implies that in order to check whether a C_4 is formed in G' it suffices to check if the updates to the modular decomposition tree produce any series node with more than one non-leaf child. In order to verify that efficiently, we introduce at each internal node N of T_G a counter, which stores the number of children of N that are not leaves. These counters can be easily maintained and checked by our dynamic modular decomposition algorithm with no increase to its time complexity. Hence, handling a vertex insertion can be supported in $O(d)$ time.

4.2.2 Deleting an Edge

Let $(a, c) \in E$ be an edge to be deleted, and let $G' = G \setminus \{(a, c)\}$. Clearly, any induced C_4 in G' is of the form $\{a, b, c, d\}$ for some vertices $b, d \in V$. By the previous discussion, in order to check whether G' contains an induced C_4, it suffices to check whether the updates to the modular decomposition tree produce any series node with a counter greater than one. By examining the updates to the tree it can be seen that the only series node whose counter might exceed one is M_{ac}, the least common ancestor of $\{a\}$ and $\{c\}$ in T_G. (Using the notation of Section 3 this happens when $|C_a| = |C_c| = 1$ and $r > 2$.) We provide below a direct proof for that.

Lemma 12 *If $\{a, b, c, d\}$ induce a C_4 in G' then $N[a] = N[c]$ in G.*

PROOF. By our assumption $(a, c) \in E$. Suppose to the contrary that $v \in V$ is adjacent to only one of a and c. Without loss of generality, suppose v is adjacent to a only. Hence, v must be adjacent to both b and d, or else G' contains an induced P_4. But then $\{d, v, b, c\}$ induce a C_4 in G, a contradiction. $\square$

Lemma 13 *If $\{a, b, c, d\}$ induce a C_4 in G', and $v \in V$ is adjacent to b or d, then v is adjacent to both a and c in G.*

PROOF. By Lemma 12, $N[a] = N[c]$ in G. Hence, it suffices to prove that v is adjacent to a. Suppose to the contrary that $(v, a) \notin E$. If $(v, b) \in E$ or $(v, d) \in E$ then $\{d, a, b, v\}$ induce a forbidden subgraph in G (either a P_4 or a C_4), a contradiction. $\square$

Let M'_{ac} be the least common ancestor of $\{a\}$ and $\{c\}$ in $T_{G'}$. Since $(a, c) \notin E(G')$, M'_{ac} is a parallel module. If M'_{ac} lies at the root of $T_{G'}$ then G' is a trivially perfect graph, since a and c are disconnected (and, therefore, cannot be part of the same induced C_4). We assume in the sequel that this is not the case.

Theorem 14 *Let P be the parent of M'_{ac} in $T_{G'}$. Then G' is a trivially perfect graph if and only if M'_{ac} is the only non-trivial maximal submodule of P.*

PROOF. Suppose to the contrary that G' is not a trivially perfect graph. Then there exist two vertices $b, d \in V$ such that $\{a, b, c, d\}$ induce a C_4 in G'. By Lemma 12, $N(a) = N(c)$ in G'. Hence, M'_{ac} is the parent of both $\{a\}$ and $\{c\}$. We claim that $M'_{ac} = \{a, c\}$. Suppose to the contrary that $v \in M'_{ac} \setminus \{a, c\}$, then v is non-adjacent to a and c (since M'_{ac} is parallel). By Lemma 13, v is non-adjacent to b and d. However, both a and c are adjacent to b and d. Hence, b must be a vertex of M'_{ac}, implying that a and c are in the same connected component in $G'(M'_{ac})$, a contradiction.

Let M'_{abcd} be the least common ancestor of M'_{ac}, $\{b\}$ and $\{d\}$ in $T_{G'}$. We now prove that $M'_{abcd} = P$. Let S_1 be a maximal submodule of M'_{abcd} such that $S_1 \supseteq M'_{ac}$. Since a is adjacent to both b and d, M'_{abcd} must be a series module. Hence, any vertex $v \in S_1 \setminus \{a, c\}$ is adjacent to b or d. By Lemma 13, v is also adjacent to a and c. Since this holds for all $v \in S_1 \setminus \{a, c\}$, and since M'_{abcd} is a series module, $S_1 = \{a, c\} = M'_{ac}$, implying that $M'_{abcd} = P$. Finally, since P is a series module, its maximal submodule that contains both b and d is a non-trivial submodule of P (different from M'_{ac}), a contradiction.

Conversely, suppose that P contains a non-trivial maximal submodule $L \neq M'_{ac}$. Since M'_{ac} is a parallel module, P is a series module. Let b and d be two non-adjacent vertices of L. Then $\{a, b, c, d\}$ induce a C_4 in G', a contradiction. $\square$

Consider the updates to T_G as a result of deleting the edge (a, c). If G' is not a trivially perfect graph, then M_{ac} was the parent of both $\{a\}$ and $\{c\}$ in T_G, and due to the update a new node $M'_{ac} = \{a, c\}$ was created and attached as a child of M_{ac}. Hence, $P = M_{ac}$ and in order to determine if G' is trivially perfect, it suffices to check the counter of M_{ac} after the update. We conclude:

Theorem 15 *There is a fully dynamic algorithm for trivially perfect graph recognition which works in $O(d)$ time per operation involving d edges.*

Acknowledgments

We thank Pavol Hell and Haim Kaplan for helpful conversations. R. Sharan was supported by a Fulbright grant. R. Shamir was supported in part by the Israel Science Foundation (grant number 565/99).

References

[1] P. Hell, R. Shamir, R. Sharan, A fully dynamic algorithm for recognizing and representing proper interval graphs, SIAM Journal on Computing 31(1) (2002) 289–305.

[2] L. Ibarra, Fully dynamic algorithms for chordal graphs, in: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99), 1999, pp. 923–924.

[3] L. Ibarra, A fully dynamic algorithm for recognizing interval graphs using the clique-separator graph, Tech. rep., University of Victoria, Victoria, Canada (2001).

[4] W.-L. Hsu, On-line recognition of interval graphs in $O(m+n \log n)$ time, Lecture Notes in Computer Science 1120 (1996) 27–38.

[5] X. Deng, P. Hell, J. Huang, Linear time representation algorithms for proper circular arc graphs and proper interval graphs, SIAM Journal on Computing 25(2) (1996) 390–403.

[6] R. M. McConnell, J. P. Spinrad, Linear-time modular decomposition and efficient transitive orientation of comparability graphs, in: Proc. Fifth Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'94), ACM Press, 1994, pp. 536–545.

[7] A. Cournier, M. Habib, A new linear algorithm for modular decomposition, in: 19th International Colloquium (CAAP'94), 1994, pp. 68–82, lNCS 787.

[8] E. Dahlhaus, J. Gustedt, R. McConnell, Efficient and practical modular decomposition, in: Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'97), 1997, pp. 26–35.

[9] J. Muller, J. Spinrad, Incremental modular decomposition, Journal of the ACM 36 (1) (1989) 1–19.

[10] D. Corneil, Y. Perl, L. Stewart, A linear recognition algorithm for cographs, SIAM Journal on Computing 14(4) (1985) 926–934.

[11] A. Brandstädt, V. B. Le, J. P. Spinrad, Graph Classes - a Survey, SIAM, Philadelphia, 1999, sIAM Monographs in Discrete Mathematics and Applications.

[12] J. Spinrad, Two dimensional partial orders, Ph.D. thesis, Dept. of Computer Science, Princeton University (1982).

[13] D. G. Corneil, H. Lerchs, L. Stewart Burlingham, Complement reducible graphs, Discrete Applied Mathematics 3 (1981) 163–174.

[14] N. Mahadev, U. Peled, Threshold Graphs and Related Topics, Elsevier, North-Holland, 1995, annals of Discrete Mathematics, Vol. 56.

[15] P. L. Hammer, B. Simeone, The splittance of a graph, Combinatorica 1 (1981) 275–284.

[16] M. C. Golumbic, Trivially perfect graphs, Discrete Math. 24 (1978) 105–107.