A Linear-time Algorithm for Proper Interval Graph Recognition

Celina M. Herrera de Figueiredo* João Meidanis† Célia Picinin de Mello†

Abstract

Interval graphs are the intersection graphs of families of intervals in the real line. If the intervals can be chosen so that no interval contains another, we obtain the subclass of proper interval graphs. In this paper, we show how to recognize proper interval graphs in linear time by constructing the clique partition from the output of a single lexicographic breadth-first search.

KEYWORDS: design of algorithms, interval graphs, indifference graphs.

1 Introduction

Interval graphs, which are the intersection graphs of families of intervals in the real line, have countless applications and have been extensively studied since their inception [4, 5]. If the intervals can be chosen so that no interval contains another, we obtain precisely the class of interval graphs that do not admit $K_{1,3}$ as an induced subgraph [9]. Such graphs are called accordingly *proper interval graphs*. The names *unit interval graph*, *indifference graph* and *time graph* are also used in the literature as synonyms of proper interval graph [4, 7, 9].

The first linear-time algorithm for interval graph recognition appeared in 1975 [1]. This algorithm uses a *lexicographic breadth-first search* (lexBFS) to find in linear time the maximal cliques of the graph and then employs special structures called PQ-trees to find an

*Universidade Federal do Rio de Janeiro, Instituto de Matemática, Caixa Postal 68530, 21944 Rio de Janeiro, RJ, Brasil. Partially supported by CNPq, grant 30 1160/91.0.

†Universidade Estadual de Campinas, Departamento de Ciência da Computação, Caixa Postal 6065, 13081 Campinas, SP, Brasil. Partially supported by FAPESP and CNPq.

ordering of the maximal cliques that characterizes interval graphs. In 1989, Korte and Möhring introduced MPQ-trees, a modified version of the original PQ-trees, and gave a simpler, incremental algorithm for interval graph recognition, which still runs in linear time [8]. In 1992, Hsu presented an alternate linear-time algorithm for interval graph recognition [6]. This algorithm does not use PQ-trees: it builds a decomposition tree that finds all interval representations for a given interval graph.

It is known that a single run of lexBFS is enough to recognize triangulated graphs [11]. Recently, K. Simon presented an intriguing algorithm, in which four iterations of lexBFS suffice to recognize interval graphs [12]. This is a hint that lexBFS may in fact be a much more powerful tool than previously thought.

None of the references cited above considers adapting an algorithm for interval graph recognition to an algorithm for proper interval graph recognition. A linear-time algorithm for proper interval graph recognition was recently given by Corneil *et al.* [2].

In this paper, we show how to recognize proper interval graphs by constructing the clique partition from the output of a single lexBFS. The algorithm runs in linear time and does not involve PQ-trees or MPQ-trees. Instead, we use a much simpler structure, similar to the one used in the lexBFS itself: a doubly-linked list of classes of nodes. Our algorithm also constructs a compact representation of all indifference orders for a proper interval graph. This representation gives a solution to the isomorphism problem for this class.

The methods used in [2] and here are different. Corneil *et al.* use three passes through the graph while we use only two passes. In addition, our algorithm gives as a by-product the clique partition of the input graph.

2 The algorithm

In this paper, G denotes an undirected, finite, connected graph. $V(G)$ and $E(G)$ are the vertex and edge sets of G, respectively. A *clique* is a set of vertices pairwise adjacent in G. A *maximal clique* of G is a clique not properly contained in any other clique. For each vertex v of a graph G, $Adj(v)$ denotes the set of vertices which are adjacent to v. In addition, $N(v)$ denotes the *neighborhood* of v, that is, $N(v) = Adj(v) \cup \{v\}$. We extend the domain of N to subsets V' of $V(G)$ by setting $N(V') = \cup_{v \in V'} N(v)$.

A *lexicographic breadth-first search* (lexBFS) is a breadth-first search procedure with the

additional rule that vertices with earlier visited neighbors are preferred. Following Korte and Möhring [8], we say that $v,\ w \in V(G)$ *disagree* on $u \in V(G)$ if exactly one of them is adjacent to u. Thus, a lexBFS produces an ordering $(v_1, \ldots, v_n)$ of $V(G)$ with the following property: if there are i, j, k with $1 \leq i < j < k \leq n$ such that $v_j,\ v_k$ disagree on v_i, then the leftmost vertex on which they disagree is adjacent to v_j.

Interval graphs are precisely those which admit a linear order on the set of maximal cliques such that the maximal cliques containing the same vertex are consecutive [3]. A clique that can appear as the first (or the last) clique in such an order is called an *outer clique*. A vertex v is *simplicial* when $N(v)$ is a clique. In addition, for interval graphs, a vertex v is *external simplicial* when $N(v)$ is an outer clique. A lexBFS in a interval graph always ends in an external simplicial vertex [8].

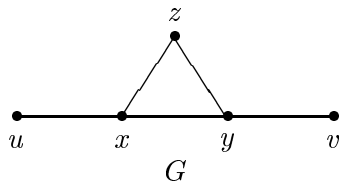
Indifference graphs too can be characterized by a linear order: their vertices can be linearly ordered so that vertices contained in the same maximal clique are consecutive [10]. We call such an order an *indifference order*. Our algorithm works by trying to construct an indifference order for the input graph.

Because a lexBFS in an indifference graph always ends in an external simplicial vertex [8], a natural idea is to remove this last vertex (call it v), perform a recursive call with the remaining graph, and then include v in the indifference order returned by the recursive call. Since v is external simplicial, there are only two possibilities: v goes either in the beginning or in the end of the returned order. If v is not adjacent to either extreme, or if the recursive call rejects, then the algorithm rejects.

Furthermore, since a lexBFS ordering without its last vertex is still a lexBFS on the graph without this vertex, a single lexBFS run can be used throughout the whole process.

Unfortunately, this simple scheme does not work because there are, in general, many indifference orders for a given graph. If the wrong one is returned, the algorithm will reject a good graph. As an example, consider the graph G depicted in Figure 1 and the given lexBFS ordering. Although there are only two indifference orders for G (one being the reverse of the other), there are four indifference orders for $G \setminus \{v\}$. Only two of them permit the placement of v beside its only neighbor y. In the other orders, y is "hidden" by the other vertices; so, the algorithm will incorrectly report that G is not an indifference graph.

To overcome this problem, throughout the whole process we work with reduced graphs. The starting point of our algorithm is the observation that for *reduced* indifference graphs



lexBFS ordering: $u\ x\ y\ z\ v$.
indifference orders for $G \setminus \{v\}$:
$u\ x\ z\ y$.
$u\ x\ y\ z$.
$y\ z\ x\ u$.
$z\ y\ x\ u$.

Figure 1: The naïve algorithm fails to recognize this indifference graph.

the indifference order is unique (except for its reverse) [10]. Two vertices v, w are *twins* when $N(v) = N(w)$. A graph is *reduced* if no two distinct vertices are twins. The *reduced graph* G' of a graph G is the graph obtained from G by collapsing each set of twins into a single vertex and removing parallel edges and loops. The "twin" relation is an equivalence relation. The partition of $V(G)$ into classes of twin vertices is called the *clique partition* of the graph.

Note that, by definition, an indifference order induces a total order on the sets of the clique partition of an indifference graph G and vice versa. This total order is an indifference order for the reduced graph G'. We shall denote by $P = (A_1, \ldots, A_l)$ the corresponding ordered clique partition. Remark that all vertices of a set A_i are twins.

Indif(G)

let $(v_1, \ldots, v_n)$ be the order produced by a lexBFS on G
initialize: $P \leftarrow (\{v_1\})$
for $i := 2$ **to** n **do**
    if v_i can be included in P **then** update P
    else reject the graph
    endif
endfor
accept the graph

Figure 2: Our lexBFS algorithm for indifference graph recognition.

The algorithm considers the vertices of G ordered according to a lexBFS. Let $(v_1, \ldots, v_n)$ be that order. By definition, the graph consisting of vertex v_1 is indifference. The algorithm tries to add each vertex of G following the given lexBFS ordering. If the current vertex can be included in the ordered clique partition of the current graph, then we continue. Otherwise the input graph is rejected. The algorithm appears in Figure 2.

We proceed to show how we can decide whether a new vertex v can be included in the current ordered clique partition P and how to update P. The following result shows that in an indifference graph all neighbors of the last vertex in a lexBFS are packed toward one end of an indifference order.

Theorem 1 *Let G be a graph and v be the last vertex visited in a lexBFS ordering on G. Suppose $H = G \setminus \{v\}$ is an indifference graph and let $P = (A_1, \ldots, A_l)$ be an ordered clique partition for H. If G is an indifference graph, then one of the following holds:*

1. *$Adj(v) \cap A_1 \neq \emptyset$. Moreover, there is an index j such that $\cup_{k=1}^{j-1} A_k \subseteq Adj(v) \subseteq \cup_{k=1}^{j} A_k$, $A_j \cap Adj(v) \neq \emptyset$, and $A_j \cap N(A_1) \neq \emptyset$.*

2. *$Adj(v) \cap A_l \neq \emptyset$. Moreover, there is an index j such that $\cup_{k=j+1}^{l} A_k \subseteq Adj(v) \subseteq \cup_{k=j}^{l} A_k$, $A_j \cap Adj(v) \neq \emptyset$, and $A_j \cap N(A_l) \neq \emptyset$.*

Proof: In an interval graph the last vertex visited in a lexBFS belongs to an outer clique [8]. If G is an indifference graph, this applies to G as well. Thus, v is an external simplicial vertex of G and therefore $Adj(v)$ is contained in an outer clique of H. Since H is an indifference graph, there are just two outer cliques, $N(A_1)$ and $N(A_l)$. Suppose $Adj(v) \subseteq N(A_1)$ and $Adj(v) \cap A_1 = \emptyset$. Let w be any element of $Adj(v)$. Since $Adj(v) \subseteq N(A_1)$, there exists $a_1 \in A_1$ such that w is adjacent to a_1. Note that, because $N(A_1)$ is a clique, any $a_1 \in A_1$ will do. Note that by supposition, a_1 is not adjacent to v. Furthermore, $Adj(w) \not\subseteq N(A_1)$ because w and a_1 are not twins. Now taking any element $z \in Adj(w) \setminus N(A_1)$ we have that $G[w, v, a_1, z]$ is isomorphic to $K_{1,3}$, contradicting the fact that G is an indifference graph. Hence, $Adj(v)$ must meet A_1 if $Adj(v) \subseteq N(A_1)$. Analogously, $Adj(v)$ must meet A_l if $Adj(v) \subseteq N(A_l)$.

So we may assume $Adj(v) \cap A_1 \neq \emptyset$. Let j be the minimum index such that $Adj(v) \subseteq \cup_{k=1}^{j} A_k$. This guarantees that $A_j \cap Adj(v) \neq \emptyset$. Because v is a simplicial vertex we have $A_j \cap N(A_1) \neq \emptyset$ and we shall now show that $\cup_{k=1}^{j-1} A_k \subseteq Adj(v)$.

If $j = 1$, this is immediate, since the left-hand side is empty. For $j > 1$, assume for a moment that the inclusion does not hold, and take $a_j \in Adj(v) \cap A_j$ and $y \in (A_1 \cup \cdots \cup A_{j-1}) \setminus Adj(v)$. Notice that a_j is adjacent to y, because both belong to the clique $N(A_1)$. However, being in distinct classes, they are not twins in H. It follows that there is either an element adjacent to y but not to a_j, or an element adjacent to a_j but not to y. The first case must be excluded since such an element would necessarily be in a class to the left of y in the indifference order, and a_j meets all classes down to A_1. Hence, there is $z \in Adj(a_j) \setminus Adj(y)$. But then $G[a_j, v, y, z]$ is isomorphic to $K_{1,3}$, a contradiction. ■

The following theorem shows how to update the clique partition. We note that this result also says that given any indifference graph, our algorithms always ends by constructing an indifference order. This implies the correctness of our algorithm because a graph is indifference if and only if it admits an indifference order.

Theorem 2 *Let G be a graph and v be the last vertex visited in a lexBFS ordering on G. Suppose $H = G \setminus \{v\}$ is an indifference graph, and let $P = (A_1, \ldots, A_l)$ be an ordered clique partition for H. Suppose further that $Adj(v) \cap A_1 \neq \emptyset$, and that there is an index j such that $\cup_{k=1}^{j-1} A_k \subseteq Adj(v) \subseteq \cup_{k=1}^{j} A_k$, $A_j \cap Adj(v) \neq \emptyset$, and $A_j \cap N(A_1) \neq \emptyset$. Then an ordered clique partition P' of G can be obtained as follows:*

1. *If $Adj(v) = \cup_{k=1}^{j} A_k$, then* $\begin{cases} P' = (A_1 \cup \{v\}, \ldots, A_l), & \text{if there is } w \text{ in } A_1 \text{ with} \\ & N(w) = N(v). \\ P' = (\{v\}, A_1, \ldots, A_l), & \text{otherwise.} \end{cases}$
2. *If $Adj(v) \neq \cup_{k=1}^{j} A_k$, then $P' = (\{v\}, A_1, \ldots, A_{j-1}, B, C, A_{j+1}, \ldots, A_l)$, where $B = A_j \cap Adj(v)$ and $C = A_j \setminus Adj(v)$.*

No other case is possible, and G is an indifference graph.

Proof: We shall prove the following: If the vertex v can be included in P, then P' is an indifference order for reduced graph G', i.e., G itself admits an indifference order and hence G is an indifference graph. Let $P = (A_1, \ldots, A_l)$ be the indifference order for reduced graph of H. Suppose $Adj(v) \cap A_1 \neq \emptyset$ and let j be as in the statement of the theorem. We examine two cases.

Case 1: $Adj(v) = \cup_{k=1}^{j} A_k$. This implies that two vertices x, y distinct from v are twins in H if and only if they are twins in G. Hence, there will be no changes in the current classes, except for the possible inclusion of v in one of them.

If there exists $w \in A_1$ such that $N(w) = N(v)$, then v and w are twins in G. Hence, $P' = (A_1 \cup \{v\}, A_2, \ldots, A_l)$ is an indifference order for G'.

If v is not twin with vertices of A_1, then it is not twin with any other vertex. Indeed, while $Adj(v) \subseteq N(A_1)$, all vertices outside A_1 have neighbors outside $N(A_1)$. For this reason, v must start a new class by itself, which should be placed near A_1 to satisfy the consecutive requirement for the maximal clique $\{v\} \cup A_1 \cup \cdots \cup A_j$ of G. Hence, $P' = (\{v\}, A_1, \ldots, A_l)$ is an indifference order for G'.

Case 2: $Adj(v) \neq \cup_{k=1}^{j} A_k$. In this case, class A_j must be split. Its elements will no longer be all twins in G, since some of them meet v while others don't. Let $B = A_j \cap Adj(v)$ and $C = A_j \setminus Adj(v)$. By hypothesis, both are nonempty. Thus $Adj(v) = A_1 \cup \cdots \cup A_{j-1} \cup B$ and $\{v\} \cup A_1 \cup \cdots \cup B$ is a maximal clique of G. The new classes B and C will replace A_j in the order, with B closer to A_1 because of v's maximal clique. Hence, $P' = (\{v\}, A_1, \ldots, B, C, A_{j+1}, \ldots, A_l)$ is an indifference order for G'. ■

3 Linear-time implementation

Let n and m be the number of vertices and the number of edges of the input graph G, respectively. The algorithm proposed above basically consists of a lexBFS followed by a loop whose body is executed at most $n - 1$ times.

It is well-known that a lexBFS can be performed in $O(n+m)$ time [11]. Thus, it remains to show how to implement the loop so that the same time bound applies. This will be done by showing that, if the input graph is an indifference graph, then each iteration consumes $O(|Adj(v)|)$, where v is the vertex tentatively added in this iteration. If the input graph is not an indifference graph, the last iteration may take $O(n)$ time. In any case, adding up for all iterations, this gives time $O(n + m)$.

We begin with the description of the structure used to represent the partition P. Each set A_r is represented by a doubly linked list of the vertices it contains, in arbitrary order. The A_r's themselves are kept in a doubly-linked list in indifference order. This is similar to the arrangement used in a lexBFS [11]. For each class A_r, a variable c_r is used to keep track of the cardinality of that class. In addition, the number of classes contained in each one of the two outer cliques is maintained. To accomplish that, we use a variable b (for *beginning*) indicating that $A_1 \cup A_2 \cup \ldots \cup A_b = N(A_1)$, the outer clique containing A_1, and a variable

e (for *end*) indicating that $A_e \cup \ldots \cup A_{l-1} \cup A_l = N(A_l)$, the outer clique containing A_l.

We are now ready to show how to implement the i-th iteration, which processes vertex $v = v_{i+1}$. Initially, all neighbors of v already in the structure are marked. This takes $O(|Adj(v)|)$ time. When a vertex is marked, a counter for marked vertices in its class is incremented. These counters must start with zero for each iteration. We will denote by m_r the value of this counter for class A_r.

All tests needed in an iteration, according to theorems 1 and 2, can be performed using the counters c_r and m_r. The test $Adj(v) \cap A_1 \neq \emptyset$ is equivalent to $m_1 \neq 0$, and can be done in constant time. For the remaining conditions, we assume that $m_1 \neq 0$. Otherwise, we check the opposite end of the structure in an analogous manner.

The index j is the maximum index for which $m_j \neq 0$. If the graph is an indifference graph, then we have $j \leq |Adj(v)|$, since all neighbors of v are packed toward one end. In any case, $j \leq n$. Thus, j is the minimum index such that $\Sigma_{k=1}^{j} m_k = |Adj(v)|$. Hence, this argument guarantees $O(n+m)$ time overall.

The condition $A_j \cap N(A_1) = \emptyset$ is equivalent to $b < j$. The condition $\cup_{k=1}^{j-1} A_k \subseteq Adj(v)$ consists in verifying whether $m_r = c_r$ for $r = 1, \ldots, j-1$. The test $Adj(v) = \cup_{k=1}^{j} A_k$ is equivalent to $m_j = c_j$. If this is true, the test "there exists $w \in A_1$ with $N(v) = N(w)$" is equivalent to $j = b$.

After all these tests have been performed, it is necessary to set $m_r \leftarrow 0$, for $r = 1, \ldots, j$, preparing for the next iteration. If v is added successfully, these are the only counters modified in this iteration. Otherwise, the loop will stop and the graph will be rejected anyway; so, we don't have to worry about future iterations.

Finally, we need to specify how the structure is updated when v is added. Adding a vertex to a class or creating a new singleton class are easily accomplished in constant time, as well as updating the cardinality counters c_r. The potential problem here is splitting a mixed class in two, one containing the marked and the other the unmarked vertices. However, this can be done with the following trick. Each time a vertex is marked, it is transferred to the beginning of the list for its class. This way, when we need to split a class A_r, we know that the first m_r elements are the ones adjacent to v. This is a convenient way of marking vertices without using an extra bit field. It also obviates the need to clear the marks: zeroing m_r automatically does this.

A last remark concerns updating the outer clique indicator b. If v is added to A_1, then

b remains the same. In all other cases, b receives the value $j+1$, reflecting the fact that the new outer clique contains the first $j+1$ classes.

References

[1] K. S. Booth and G. S. Lueker, Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms, *J. Comput. System Sci.* **13** (1976) 335–379.

[2] D. G. Corneil, H. Kim, S. Natarajan, S. Olariu, A. P. Sprague, Simple linear time recognition of unit interval graphs, to appear in *Inf. Proc. Letters* (1995).

[3] P. C. Gilmore and A. J. Hoffman, A characterization of comparability graphs and interval graphs, *Canad. J. Math.* **16** (1964) 539–548.

[4] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, N.Y., 1980.

[5] G. Hajós, Über eine Art von Graphen, *Internationale mathematische nachrichten* **11** (1957) problem 65.

[6] W. L. Hsu, A simple test for interval graphs, *Graph-Theoretic Concepts in Computer Science, International Workshop WG* (1992).

[7] B. Hedman, Clique graphs of time graphs, *J. Comb. Theory B* **37** (1984) 270–278.

[8] N. Korte and H. Möhring, An incremental linear-time algorithm for recognizing interval graphs, *SIAM J. Comput.* **18** (1989) 68–81.

[9] F. S. Roberts, Indifference graphs, A characterization of comparability graphs and of interval graphs, in: F. Harary (ed.), *Proof Techniques in Graph Theory*, 139–146, (Academic Press, New York, 1969).

[10] F. S. Roberts, On the compatibility between a graph and a simple order, *J. Comb. Theory B* **11** (1971) 28–38.

[11] D. J. Rose, R. E. Tarjan and G. S. Lueker, Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.* **5** (1976) 266–283.

[12] K. Simon, A new simple linear algorithm to recognize interval graphs, *Computational Geometry–Methods, Algorithms and Applications, International Workshop on Computational Geometry CG '91 - LNCS 553* (1991) 289–308.