Logarithmic Time Parallel Algorithms for Comparability and Interval Graphs

Mark B. Novick*

TR 89-1015
June 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This work was supported by NSF grant CCS-8806979.

Logarithmic time parallel algorithms for recognizing comparability and interval graphs

Mark B. Novick *

Abstract

We give fast parallel algorithms for recognizing and representing comparability graphs, the graphs that can be transitively oriented, and interval graphs, the intersection graphs of intervals along the real line. Under the CRCW PRAM model, both algorithms use $O(n^3)$ processors in $O(\log n)$ time to check if a graph belongs to the desired class, and if it does then a valid representation of the graph can be produced. The algorithms gain their efficiency by using fast algorithms for finding the modular decomposition of a graph. Both problems were known to be in NC, but the known algorithms require more time than ours does.

Keywords: comparability graphs, inteval graphs, analysis of algorithms, parallel processing

1 Introduction

Interval and comparability graphs are useful models in a variety of applications (see [9]), particularly scheduling problems. Both polynomial-time and NC algorithms are known for recognizing these graphs, and representing them in terms of partial orders. Kozen *et al.* [8] and Helmbold and Mayr [6] gave parallel algorithms for solving these problems on comparability graphs. Their algorithms required at least $\Omega(n^4)$ processors and $\Omega(\log^2 n)$ time. We improve these bounds to $O(n^3)$ processors and $O(\log n)$ time on a CRCW-PRAM.

Klein [7] designed an $O(m)$ processor, $O(\log^2 n)$ time algorithm for recognizing and representing interval graphs. His algorithm is an efficient parallelization of Booth and Lueker's sequential algorithm [1]. Our algorithm uses $O(\log n)$ time and $O(n^3)$ processors. We improve upon the running time of Klein's algorithm at the cost of using more processors. His method takes advantage of the fact that interval graphs are also chordal graphs, graphs with no induced cycle of length more than three. On the other hand, our method is based on the fact that the complement of an interval graph is a comparability graph. Previous algorithms based on this approach [4,8] have been inefficient in comparison to chordal graph-based methods, but we give a sequential algorithm for the problem that runs in $O(n^2)$ time, nearly as fast as Booth and Lueker's algorithm.

A key step in our method is an efficient algorithm for finding the modular decomposition

*Computer Science Department, Cornell University, Ithaca, NY 14853-7501. Supported by NSF grant CCS-8806979.

of a graph. This modular decomposition is also useful in succintly representing all the valid representations of an interval or comparability graph.

2 Definitions

We let $G = (V, E)$ denote a graph in the usual way with $n = |V|$ and $m = |E|$. E. The complementary graph of G, G^c, contains as edges E^c all edges not present in G. Many definitions and results below are symmetric in E and E^c; in the statement of such results, we use the letter F to represent either E or E^c, and F^c to represent the other.

The notion of a transitive orientation is important for both comparability and interval graphs.

Definition 2.1 *A transitive orientation of G is a subset T of E such that*

1. *for each $uv \in E$, either $uv \in T$ or $vu \in T$ but not both;*
2. *if $uv, vw \in T$, then $uw \in T$.*

The graph G is called a comparability graph if there exists a transitive orientation of it.

Definition 2.2 *Graph G is an interval graph if there is a collection $\mathcal{I}$ of intervals along the real line where each interval corresponds to a vertex of G such that $I_u \cap I_v \neq \emptyset$ if and only if u and v are adjacent vertices in G.*

We will find it useful to perform a modular decomposition when recognizing comparability and interval graphs.

Definition 2.3 *A module is a set A of vertices that are indistinguishable by the vertices outside the set: i.e., any vertex in $V - A$ is either adjacent to all vertices of A or to none of them.*

Modules are formed from smaller submodules in three ways: parallel, series, and neighborhood modules.

Definition 2.4 *Parallel modules are the disjoint union of a collection of submodules. Series modules are the disjoint union of a collection of submodules, plus edges between every pair of vertices in different modules is joined by an edge. In a neighborhood module, the graph induced by the submodules is both connected and complement-connected.*

Definition 2.5 *A modular decomposition of G is a tree whose nodes are modules of G. Every vertex of G is a leaf in this decomposition tree. A parent node in this tree contains all of its children as submodules. The children of a parallel module form a maximal set of connected submodules. The submodules of a series module form a maximal set of complement-connected graphs. The children of a neighborhood module are maximal proper submodules of the neighborhood module.*

The modular decomposition of a comparability graph is called the Gallai decomposition, after Tibor Gallai [3]. Booth and Lueker's PQ-tree representation (see [1]) mostly consists of a modular decomposition of interval graphs. The Gallai decomposition and PQ-tree succintly provide all the possible representations of comparability and interval graphs, respectively.

3 Comparability graph recognition

Our algorithm for comparability graph recognition and orientation is a more efficient version of the one proposed by Kozen, Vazirani, and Vazirani [8]. Their algorithm first finds the Gallai decomposition, and then transitively orders the graph. We give an algorithm that can find the modular decomposition of a graph on a concurrent-read concurrent-write parallel random-access machine (CRCW PRAM with arbitrary resolution of write conflicts) in $O(\log n)$ time using $O(n^3)$ processors. Several researchers (see [10]) have shown that any graph has a unique modular decomposition

In the following definition, directing uv in one direction forces us to direct $u'v'$ in the "same" direction if we are to obtain a transitive ordering of G.

Definition 3.1 *For $uv, u'v' \in E$, define $uv \,\Gamma\, u'v'$ if either*

1. *$u = u'$ and $vv' \in E^c$, or*
2. *$v = v'$ and $uu' \in E^c$.*

Let Γ^* be the reflexive transitive closure of Γ. The Γ^*-class of uv is denoted $[uv]$ and is called the implication class of uv. The set $[uv] \cup [vu]$ is called the color class of uv. The set of vertices touched by edges of $[uv]$ is denoted by $V[uv]$. Note that Γ and Γ^* are defined on edges in E^c as well as those in E.

Definition 3.2 *For $uv, u'v' \in E$, define $uv \,\Delta\, u'v'$ if either*

1. *$u = u'$ and $vv' \notin [vu] \cup [uv']$, or*
2. *$v = v'$ and $uu' \notin [uv] \cup [vu']$.*

Let Δ^* be the reflexive transitive closure of Δ. The Δ^*-class of uv is denoted by $[\![uv]\!]$. The set of vertices touched by edges of $[\![uv]\!]$ is denoted $V[\![uv]\!]$.

Definition 3.3 *A subset $A \subseteq V$ is called normal if for all $u \notin A$ and $v, v' \in A$, $[\![uv]\!] = [\![u'v']\!]$.*

Normal sets and modules are related by the following lemma.

Lemma 3.1 *Every normal set of G also forms a module of G, though not necessarily a series, parallel, or neighborhood module.*

Two lemmas of Kozen *et al.* are useful in characterizing normal sets.

Lemma 3.2 *$V[uv]$ is the smallest normal set containing u and v.*

Lemma 3.3 *If A, B are normal, then either $A \subseteq B$, $B \subseteq A$, or $A \cap B = \emptyset$.*

By using these lemmas, they had characterized the normal sets of a graph, and shown when a graph has no nontrivial normal subsets.

Theorem 3.1 *The following are all the normal subsets of G:*

1. singletons $\{u\}$, $u \in V$;

2. $V \setminus [uv]$, $u, v \in V$.

Theorem 3.2 *A graph G has no nontrivial normal subsets if and only if one of the following conditions holds:*

1. G is a clique;

2. G is totally disconnected;

3. G contains at least four vertices and has exactly two color classes, one each in E and E^c.

These three cases correspond to series, parallel, and neighborhood modules respectively. Kozen, Vazirani, and Vazirani gave the following algorithm as part of an algorithm to orient comparability graphs. In fact their algorithm can be used to compute the modular decomposition of any graph. The steps of the algorithm are:

1. Determine all classes $[uv]$ and $[uv]$ for all pairs of vertices u and v.

2. Find all normal sets.

3. Create the tree of normal sets, ordered by inclusion.

We give a more efficient implementation of their algorithm.

Theorem 3.3 *The modular decomposition of a graph can be found in $O(\log n)$ time on a CRCW-PRAM with $O(n^3)$ processors.*

Proof: The forcing relations [illegible] and [illegible] are symmetric, so we find their transitive closure by graph connectivity algorithms [11]. This graph has $O(n^3)$ edges because each uv is only adjacent to vertices which contain u and v. We can check two normal sets for inclusion by checking if they contain a vertex in common, and which set has larger size. By a previous lemma we know that two normal sets cannot just partially intersect. Either one of them must contain the other, or they are disjoint. The bottleneck in the algorithm is the first step, which uses $O(n^3)$ processors and $O(\log n)$ time. ∎

The process of finding the modular decomposition of a comparability graph is the dominant one in the parallel algorithms for orienting a comparability graph. Applying our algorithm also gives a more efficient algorithm for orienting a graph than the one given by Helmbold and Mayr [6] did. Their algorithm required the use of $O(n^4)$ processors because they found a maximal independent set during their algorithm. More efficient algorithms are now known for finding maximal independent sets [5] so Helmbold and Mayr's algorithm can be made to run with only $O(n^3)$ processors, but even these improved versions require at least $O(\log^3 n)$ time.

4 Interval graph recognition

We can check if a graph is an interval graph by using a result of Gilmore and Hoffman [4]:

Lemma 4.1 *A graph G is an interval graph if and only if it does not contain an induced cycle of length four and its complement is a comparability graph.*

A special type of partial order arises when we transitively order the complement of an interval graph.

Definition 4.1 *An interval order $\prec$ of V is a partial order satisfying:*

$$a \prec x, b \prec y \Rightarrow a \prec y \vee b \prec x$$

for all $a, b, x, y \in V$.

Fishburn (Theorem 2.6 of [2]) has shown that every interval order has an interval representation:

Lemma 4.2 *$(V, \prec)$ is an interval order if and only if there is a mapping F from V into closed intervals along the real line such that for all $x, y \in V$, $x \prec y$ if and only if the right endpoint of x's interval is less than the left endpoint of y's interval.*

The transitive orientation of the complement of an interval graph must be an interval order, else the interval graph itself would contain and induced four-cycle.

With respect to an interval representation we introduce the following notation:

Definition 4.2 *We say $x \prec^{-} y$ if the left endpoint of x's interval is less than the left endpoint of y's interval. Likewise we say $x \prec^{+} y$ if the same statement is true of the right interval endpoints.*

Fishburn (Theorem 2.5 of [2]) proved another useful result:

Lemma 4.3 *If $(V, \prec)$ is an interval order, then the relation $\equiv^{-}$ partitions V into the left congruence classes $A_1, \ldots, A_m$. Likewise, $\equiv^{+}$ partitions V into the same number of right congruence classes $B_1, \ldots, B_m$. Furthermore, if each of these partitions is sorted in increasing order, then for $1 \le i \le m$ all the left endpoints of vertices in A_i precede all the right endpoints of vertices in B_i, yet for $1 \le i < m$ all the right endpoints of vertices in B_i precede all the left endpoints of vertices in A_{i+1}.*

Instead of computing these congruence classes by just comparing left endpoints with left endpoints, and right ones with right ones, we can just check the number of vertices which precede and follow a given vertex in the interval order.

Theorem 4.1 *If $(V, \prec)$ is an interval order, then vertices x and y will be in the same left congruence class if and only if every vertex v that is less than x is also less than y, and vice versa. Similarly, x and y will be in the same right congruence class if and only if x and y are both less than the same set of vertices.*

Proof: If x and y are in the same left congruence class, then by the previous lemma they are greater than the same set of right congruence classes. If they belonged to different left congruence classes, then the larger of the two vertices, say x, would be greater than some right congruence class that y is not greater than. The proof of the other half of the theorem proceeds symmetrically. ∎

This theorem yields a constructive way of finding an interval representation corresponding to a given interval order. We can even more quickly compute the congruence classes by noting that to find x's left congruence class we only have to count the number of vertices less than x, rather than actually maintaining this set.

The algorithm for interval graph recognition and representation consists of the following steps:

1. Check if the complement of G is a comparability graph. If not, then G is not an interval graph. Otherwise, transitively orient the edges of the complement.
2. Compute the left and right congruence classes by counting the number of vertices less than and greater than each vertex.
3. Construct the interval representation from the equivalence classes.
4. Find the interval corresponding to this representation. Check to see if it is the same as G. If not, then G is not an interval graph because it contains an induced four-cycle.

Theorem 4.2 *The interval graph recognition and representation algorithm runs in $O(\log n)$ time using $O(n^3)$ processors on a CRCW-PRAM.*

Proof: The first step uses $O(\log n)$ time and $O(n^3)$ processors. The congruence classes can be found with only $O(m)$ processors, as can the interval representation. Checking that the interval representation is valid requires $O(n^2)$ processors if we compare adjacency matrices. If instead we first check to see if the number of edges is the same in both graphs, and later compare edges, then we only need $O(m)$ processors. The performance of the algorithm is dominated by the calculations of the first step. ▮

We can also get an $O(n^2)$ time sequential version of this algorithm for interval graph recognition. There are faster sequential algorithms known for interval graph recognition, *e.g.* [1], but none of these algorithms is based on the fact that the complement of an interval graph is a comparability graph. Spinrad [12] has shown that the edges of a graph can be oriented in $O(n^2)$ time such that this orientation is a valid orientation if and only if the graph is a comparability graph. Since our check to see if the graph really is an interval graph also works in $O(m)$ time, we can do the entire recognition and representation in quadratic time.

5 An example

We give an example showing these two algorithms at work on the graph in Figure 1.

The normal sets of this graph are:

$$\begin{array}{lll} V[uv] = \{a,b,c,d\} & \text{when } u,v \in \{a,b,c,d\}, u \neq v \\ V[ue] = \{a,b,c,d,e\} & \text{when } u \in \{a,b,c,d\} \\ V[uv] = \{a,b,c,d,e,f,g\} & \text{when } u \in \{f,g\} \end{array}$$

One possible way of ordering the complement is shown in Figure 2. Given this ordering we compute the number of vertices which precede and follow each vertex.

	a	b	c	d	e	f	g
precede	1	1	2	3	1	0	6
follow	3	2	1	1	1	6	0

Only three values appear in both the **precede** and **follow** lists. Therefore, this interval graph has exactly three maximal cliques. The interval representation corresponding to this information appears in Figure 2.

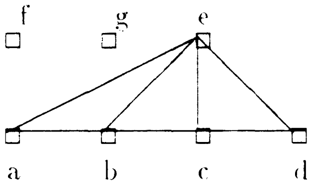

Figure 1: Example interval graph

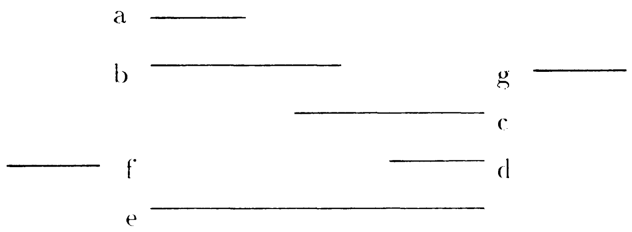

Figure 2: Its interval representation

References

[1] Kellogg S. Booth and George S. Leuker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Sys. Sci.*, 13:335–379, 1976.

[2] Peter C. Fishburn. *Interval orders and interval graphs*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons, New York, 1985.

[3] Tibor Gallai. Transitiv orientierbare graphen. *Acta Math. Acad. Sci. Hungar.*, 18:25–66, 1967.

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

[5] Mark Goldberg and Thomas Spencer. A new parallel algorithm for the maximal independent set problem. In *28th FOCS*, pages 161–165, 1987.

[6] David Helmbold and Ernst Mayr. Perfect graphs and parallel algorithms. In *1986 International Conference on Parallel Processing*, pages 853–860. IEEE, 1986.

[7] Philip Klein. Efficient parallel algorithms for chordal graphs. In *29th FOCS*, pages 150–161, 1988.

[8] Dexter Kozen, Umesh V. Vazirani, and Vijay V. Vazirani. NC algorithms for comparability graphs, interval graphs, and unique perfect matchings. Technical Report TR 86-799, Department of Computer Science, Cornell University, December 1986.

[9] Rolf H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In Ivan Rival, editor, *Graphs and Order*, pages 41–101. D. Reidel, 1984.

[10] John H. Muller and Jeremy Spinrad. Incremental modular decompostion. *J. of Assoc. Comput. Mach.*, 36:1–19, 1989.

[11] Y. Shiloach and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *J. of Algorithms*, 3:57–63, 1982.

[12] Jeremy Spinrad. On comparability and permutation graphs. *SIAM J. Comput.*, 14:658–670, 1985.