The Ultimate Interval Graph Recognition Algorithm?
(Extended Abstract)

Derek G. Corneil* Stephan Olariu† Lorna Stewart‡

Summary of Results

An independent set of three vertices is called an *asteroidal triple* if between every two vertices in the triple there exists a path avoiding the neighbourhood of the third. A graph is asteroidal triple-free (AT-free, for short) if it contains no asteroidal triple. A classic result states that a graph is an interval graph if and only if it is chordal and AT-free. Our main contribution is to exhibit a very simple, linear-time, recognition algorithm for interval graphs involving four sweeps of the well-known Lexicographic Breadth First Search. Unlike the vast majority of existing algorithms, we do not use maximal cliques in our algorithm – we rely, instead, on a less well-known characterization by a linear order of the vertices.

1 Introduction

Interval graphs arise naturally in the process of modeling real-life situations, especially those involving time dependencies or other restrictions that are linear in nature. The literature contains dozens of papers describing applications of interval graphs to such diverse areas as archaeology, biology, psychology, sociology, management, genetics, engineering, scheduling, transportation and others. For a wealth of information about interval graphs the interested reader is referred to [5], where many of the above applications are summarized.

Early characterizations of interval graphs in terms of forbidden configurations appear in Lekkerkerker and Boland [9] and Gilmore and Hoffman [4]. In particular, the following characterization offered in [4] turned out to be the workhorse of the vast majority of recognition algorithms for interval graphs.

Theorem 1.1. [4] A graph is an interval graph if and only if its maximal cliques can be linearly ordered in such a way that for every vertex in the graph the maximal cliques to which it belongs occur consecutively in the linear order. □

*Department of Computer Science, University of Toronto, Toronto, Ontario, Canada. Supported by a grant from NSERC. (dgc@cs.toronto.edu)

†Department of Computer Science, Old Dominion University, Norfolk, Virginia, U.S.A. Supported, in part, by NSF grant CCR-9407180 and ONR grant N00014-95-1-0779. (olariu@cs.odu.edu)

‡Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada. Supported by a grant from NSERC. (stewart@cs.ualberta.ca)

A crude implementation of the characterization in Theorem 1.1 yields a recognition algorithm that runs in time proportional to the cube of the number of vertices in the graph. Later, Booth and Lueker [1] used PQ-trees to compute the required linear order on the set of maximal cliques, if such an order exists. In the process, they reduced the complexity to linear in the size of the graph, which is best possible. In spite of this, the algorithm of [1] was less than perfect. For one thing, PQ-trees are complicated data structures and, not surprisingly, the resulting algorithm was rather involved. Roughly ten years later, Korte and Möhring [8] streamlined the recognition algorithm of Booth and Lueker, using a variant of PQ-trees that they call MPQ-trees, while still using the characterization by maximal cliques as the focal point of their algorithm. Recently, Hsu [7] demonstrated that PQ-trees are not essential for recognizing interval graphs by using modular decomposition techniques and not relying on Theorem 1.1. Although still linear time, this algorithm is also difficult to implement since there is no known easy linear time algorithm to perform modular decomposition. More recently, Habib, Paul and Viennot [6] have developed a linear time algorithm that is easy to implement. Their algorithm uses Lexicographic Breadth First Search to determine that the given graph is chordal and to produce a clique tree. They then manipulate this clique tree into a clique path (if the graph is interval), thus satisfying Theorem 1.1.

Instead of using Theorem 1.1, our algorithm is based on the following characterization of interval graphs by Olariu [11].

Theorem 1.2. [11] A graph $G = (V, E)$ is an interval graph if and only if there exists a linear order $\prec$ on the set of its vertices such that for every choice of vertices u, v, w, with $u \prec v$ and $v \prec w$,

$$uw \in E \text{ implies } uv \in E. \ \square \tag{1}$$

The main contribution of our paper is to propose a very simple recognition algorithm for interval graphs. Our idea is to show that the well-known Lexicographic Breadth First Search (LBFS) devised by Rose, Tarjan, and Lueker [13] to recognize chordal graphs, when modified to break ties in a specific way and used in a quadruple sweep, yields a linear order on the set of vertices that satisfies condition (1) of Theorem 1.2 if, and only if, the graph under scrutiny is an interval graph. Furthermore our algorithm does not work on the cliques of the given graph, instead all operations are performed on the vertices of the graph.

Our recognition algorithm was motivated, in part, by our previous work on a larger class of graphs, namely *asteroidal triple-free (AT-free) graphs*. An *asteroidal triple* is an independent triple of vertices such that between each pair of vertices in the triple there is a path that avoids the neighbourhood of the third vertex. In particular, we borrow from [2, 3] most of the terminology used in this paper, along with a number of deep results that have a direct impact on our recognition algorithm. The connection between AT-free graphs and interval graphs is captured in the following classical result of Lekkerkerker and Boland [9].

Theorem 1.3. [9] A graph is an interval graph if and only if it is chordal and asteroidal triple-free. □

Thus our algorithm may be viewed as the first interval graph recognition algorithm that brings the rich structural results of AT-free graphs to bear on interval graphs.

The remainder of this work is organized as follows: Section 2 reviews graph-theoretic concepts, establishes notation and terminology and introduces LBFS; Section 3 presents our algorithm together with an overview of the implementation issues and an example; Section 4 lists some of the theoretical results used in the proof of correctness.

2 Preliminaries

All the graphs in this work are finite with no loops or multiple edges and unless explicitly stated otherwise are assumed to be connected. We refer to a path joining vertices x and y as an x, y-path. Unless stated otherwise, we assume that the paths are induced. A vertex u *intercepts a path* π if u is adjacent to at least one vertex on π; otherwise, u is said to *miss* π. For vertices u, v in G, we let $D(u, v)$ denote the set of vertices that intercept all u, v-paths. (u, v) is said to be a *dominating pair* whenever $D(u, v) = V$. We say that vertices u and v are *unrelated with respect to* y if $u \notin D(v, y)$ and $v \notin D(u, y)$. A vertex y of G is said to be *admissible* if there are no vertices in G unrelated with respect to y. A vertex v is said to be *simplicial* if its neighbours are pairwise adjacent. Finally a linear order $\prec$ that satisfies condition (1) of Theorem 1.2 is said to be *umbrella free*; furthermore an edge uw is called an *umbrella* if there is a v such that $u \prec v \prec w$ and $uv \notin E$.

We now turn to the well-studied Lexicographic Breadth First Search (LBFS) introduced by Rose, Tarjan and Lueker [13]. In this fundamental paper they showed that a given graph $G = (V, E)$ is chordal if and only if the ordering of V produced by an arbitrary LBFS is a perfect elimination scheme (ie no vertex forms the center of a P_3 with any previously visited vertices). In light of Theorem 1.3, it is interesting to note that LBFS also plays a critical role in the algorithmic study of AT-free graphs. In particular, in [3] we showed that a double sweep LBFS can be used to find (in linear time) a dominating pair of a connected AT-free graph. These two seemingly disparate uses of LBFS and Theorem 1.3 give a strong hint that clever use of LBFS can be used to recognize interval graphs as well as to reveal deep structural results on these graphs.

Before presenting our algorithm, we reproduce the details of a variant of LBFS [13] for u a vertex of a connected graph $G = (V, E)$. Note that this version of LBFS allows arbitrary tie-breaking; later in the paper we impose specific tie-breaking mechanisms.

Procedure LBFS(G, u):
{Input: a connected graph $G = (V, E)$ and a distinguished vertex u of G;
Output: an ordering σ of the vertices of G}
begin
label$(u) \leftarrow |V|$;
for each vertex v in $V - \{u\}$ **do**
label$(v) \leftarrow \Lambda$;
for $i \leftarrow |V|$ **downto** 1 **do begin**
pick an unnumbered vertex v with lexicographically the largest label; $(\star)$
$\sigma(v) \leftarrow |V| + 1 - i$; {assign number $|V| + 1 - i$ to v }
for each unnumbered vertex w in $N(v)$ **do**
append i to label(w)
end
end; {LBFS}

As mentioned above, this generic LBFS algorithm allows arbitrary choice of a vertex in step $(\star)$. We call the set of tied vertices encountered in step $(\star)$ a *sub*slice* and denote it by S. We now describe two LBFS algorithms that deterministically choose the vertex v. In both algorithms the choice of v depends on either one or two previous LBFS sweeps.

Procedure LBFS$^+$ (G, u):
For this LBFS procedure, one previous sweep is needed. Now v is chosen to be the vertex in S that appears *last* in the previous sweep.
Note: This version of LBFS was used by Simon [12] in his interval graph recognition algorithm. In particular, he performed an arbitrary LBFS followed by three applications of LBFS$^+$. He claimed that for an arbitrary interval graph G and an arbitrary LBFS of it, the ordering resulting from the fourth sweep would exhibit a linear ordering of the cliques of G. Ma [10] however, showed that this algorithm is flawed and that for any constant c, there is an interval graph and an initial LBFS such that after c sweeps of LBFS+, the linear ordering of cliques is still not apparent!

Procedure LBFS* (G, u):
This variant of LBFS needs *two* previous LBFS sweeps. Given a sub*slice S (ie as identified in step $(\star)$ of the algorithm), we select two vertices a and b where a is the last S- vertex in the first sweep and b is the last S-vertex in the second sweep. To choose between a and b we introduce some terminology.

We say that vertex x in S *flies* (F) if there is a vertex y such that y occurs after S and $xy \in E$. Further x is said to *neighbour fly* (NF) if it does not fly but it has a neighbour in S that flies. Finally, vertex x is said to be *OK* if it neither flies nor neighbour flies. LBFS* chooses between a and b by referring to the following decision table.

		b		
		F	NF	OK
	F	b	b	b
a	NF	a	b	b
	OK	a	a	b

For example, if a neighbour flies and b flies, then a is chosen. At first glance, it seems surprising that LBFS* could have an easy linear time implementation. We now discuss the full algorithm as well as such implementation issues.

3 The Algorithm

Our presentation of the 4-sweep LBFS interval graph recognition algorithm also shows how elementary data structuring techniques can be used to ensure a linear time implementation. For an arbitrary LBFS ordering τ and vertex v, we let the *index*, $i_\tau(v)$ denote the largest τ value of all vertices adjacent to v. Thus $i_\tau(v) > \tau(v)$ if and only if v is adjacent to some vertex that occurs after it in the sweep τ. For convenience we set $i_\tau(v) = \tau(v)$ if v is not adjacent to any vertex that occurs after it in the sweep τ.

Step 1. LBFS from an arbitrary vertex x and let y be the last vertex visited. Let σ denote this ordering.

Step 2. Using σ, LBFS$^+$ from y and let z be the last vertex visited. Let σ_+ denote this ordering.

Step 3. For every vertex v in G, calculate its *index*, $i_+(v)$ and its A set where $A(v)$ is the set of neighbours of v that are before v in σ_+ and are adjacent to a vertex after v in σ_+ (ie the i_+ value is greater than $\sigma_+(v)$). Note that the elements of these sets are linked so that a particular element may be removed from all A sets in O(the number of occurrences of the element). We also store the cardinalities of these sets and update these cardinalities every time a vertex is removed.

Step 4. Using σ_+, LBFS$^+$ from z. This sweep will end at y. Let σ_{++} denote this ordering.

Step 5. For every vertex v in G, calculate its *index* $i_{++}(v)$ and its B set where $B(v)$ is the set of neighbours of v that are before v in σ_{++} and are adjacent to a vertex after v in σ_{++}. Again the elements are linked to allow fast deletion and again the cardinalities are stored.

Step 6. Using σ_+ and σ_{++}, LBFS* from y. Let σ_* denote this ordering. As each vertex is visited in this LBFS, it is removed from all A and B sets and the cardinalities of these sets are updated.

The cost to do the removal of vertex v is $O(\deg(v))$. We now indicate how ties are to be broken in this sweep. Assume that sub*slice S has been identified with a the last S-vertex in σ_+ and b the last S-vertex in σ_{++}.

(a) If $i_+(a) > \sigma_+(a)$, CHOOSE b.

(b) If $i_{++}(b) > \sigma_{++}(b)$, CHOOSE a.

(c) If $|B(b)| = 0$ OR $|A(a)| \neq 0$, CHOOSE b.

(d) Let y be an arbitrary element of $B(b)$. If $i_+(y) = \sigma_+(a)$, then CHOOSE b, else CHOOSE a.

Note that since the cardinalities are stored, all steps can be done in $O(1)$ time.

Step 7. If σ_* is umbrella free, G is an interval graph; else G is not interval.

We now illustrate this algorithm by means of an example. Consider the graph in Figure 1.

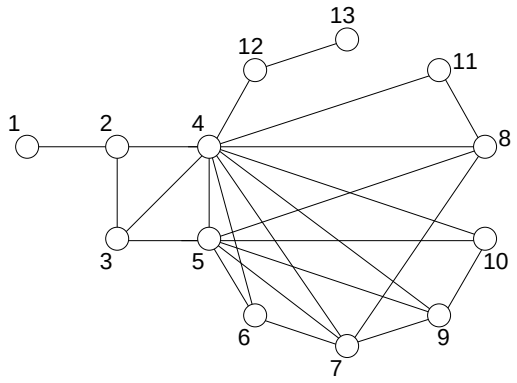

Figure 1: Illustrating the interval graph recognition algorithm

Our algorithm will produce the following sweeps when given the initial sweep σ:

σ: 6 7 4 5 8 9 10 3 11 2 12 1 13
σ_+: 13 12 4 2 3 5 10 9 7 8 6 11 1
σ_{++}: 1 2 3 4 5 6 7 8 9 10 11 12 13
σ_*: 13 12 4 11 8 7 5 6 9 10 3 2 1

There are a few interesting facts illustrated by this example. First in the final sweep, when $S = \{6, 9\}$ is encountered, $a = 6$ and $b = 9$. Since b flies and a is OK, a is chosen. If an ordinary LBFS$^+$ were used here, the b vertex would have been chosen and there would have been a resulting umbrella 9,10 over 6. Thus, in addition to the examples produced by Ma [10], this example shows how the Simon 4-sweep algorithm fails. Secondly, since LBFS* only needs two previous sweeps, it is natural to wonder whether a 3-sweep algorithm would suffice (ie LBFS, LBFS$^+$, LBFS*). In fact the example shows that this simpler algorithm would also fail. Given the same starting sweep σ, the third sweep (even though an LBFS* sweep) would be exactly the same as the σ_{++} sweep shown

above with its umbrella of 8,11 over both 9 and 10.

Apart from being umbrella free, the final LBFS of an interval graph has some other interesting properties that we now discuss.

First, directly from this final sweep we can construct an interval representation of G by representing vertex v by the interval $[\sigma_*(v), i_*(v)]$. (Recall that $i_*(v) = \sigma_*(v)$ if v is not adjacent to any vertices that follow it in σ_*.) The interval representation for the graph in Figure 1 is presented in Figure 2. Note that we have slightly extended the intervals to make the intersections more obvious.

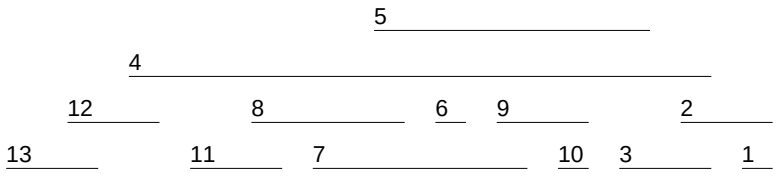

Figure 2: Interval representation for the graph in Figure 1

Secondly, either from such an interval representation, or directly from the final sweep, we can easily construct a linear ordering of the maximal cliques of G (ie an ordering that satisfies Theorem 1.1). To construct such a set from the interval representation, sweep from left to right. For each position determine the intervals that overlap this position; this set clearly forms a clique, although it may not be maximal. To ensure that we only keep maximal cliques, the clique at position i is rejected if it is strictly included in the clique at position $i + 1$. For our example the set of cliques in linear order are:

13,12
12,4
4,11,8
4,8,7,5
4,7,5,6
4,7,5,9
4,5,9,10
4,5,3
4,3,2
2,1

We now turn to the correctness of the algorithm.

4 Correctness

The proof of the correctness of the algorithm is quite long and complicated. It does however, reveal some interesting structural results on interval graphs and their behaviour in a LBFS ordering. In the following we state some of the more interesting such results, especially ones that directly impact the algorithm. The full details and proofs are in the journal version of the paper, currently in preparation.

Both LBFS$^+$ and LBFS* choose a vertex of sub*slice S that is the last S-vertex in some previous LBFS ordering. Somewhat surprisingly, the motivation for doing this comes from results on both chordal and AT-free graphs. We begin by stating a result that is implicit in [13].

Lemma 4.1. [13] Let v be an arbitrary vertex of an interval graph G, and let W denote the set of vertices w with $\sigma(w) \leq \sigma(v)$ for arbitrary LBFS ordering σ. Vertex v is simplicial in the subgraph of G induced by W. □

The corresponding result for AT-free graphs is:

Lemma 4.2. [3] Let v be an arbitrary vertex of an AT-free graph G and let W denote the set of vertices w with $\sigma(w) \leq \sigma(v)$ for arbitrary LBFS ordering σ. Vertex v is admissible in the subgraph of G induced by W. □

Thus if our graph is an interval graph, any vertex chosen at step ($\star$) by either LBFS$^+$ or LBFS* has the property that it is both simplicial and admissible. We call such vertices *good* vertices. The following lemma relates good vertices with the ordering of the vertex set imposed by an LBFS.

Lemma 4.3. A vertex w in an interval graph G is good if and only if there is some LBFS ordering τ of G such that w is the last vertex in τ. □

We say that an LBFS ordering τ is *good* if every sub*slice of τ starts and ends with vertices that are good for that sub*slice. Clearly all of our σ_+, σ_{++} and σ_* orderings are good although σ itself may not be. Indeed, as illustrated by the example in Figure 1, it is precisely this fact that led to the counter-example for the proposed 3-sweep algorithm (LBFS, LBFS$^+$, LBFS*).

The decision table that "drives" LBFS* is based on the notions of flying and neighbour flying. The reason for this is captured by the following lemma.

Lemma 4.4. Let C be a connected component of a sub*slice S of G wrt good LBFS τ. If $w \in C$ is a flyer, then w is either a good vertex of S or is adjacent to a good vertex of S. □

Before turning to the algorithm itself, we state a lemma that shows that S, a sub*slice for some LBFS is "well-behaved" in all other LBFSs of the graph.

Lemma 4.5. Let G be a chordal graph and let S be a sub*slice of an arbitrary LBFS ordering τ of G. Further let σ be any other LBFS ordering of the vertex set of G. Then the restriction of σ to S is a legitimate LBFS ordering of the graph induced on the vertices of S. □

To put this lemma in perspective, it is important to note that the desired property does not hold for arbitrary subsets of chordal (or even interval) graphs G. For example consider the interval graph

shown in Figure 3. The numbering of the vertices indicates a legitimate LBFS ordering; however when vertex 1 is removed the restriction of this ordering to the remaining subset is not a legitimate LBFS ordering of the subset.

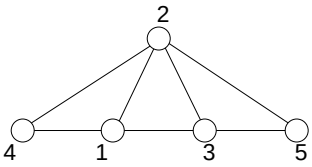

Figure 3: Putting Lemma 4.5 in perspective

Finally we turn to the algorithm itself. In particular we look at Step 6 and how it relates to the decision table that controls the choice of vertex in LBFS*. To justify how steps (a) and (b) correspond to the first row and column of the decision table we note.

Lemma 4.6. Let G be an interval graph and let S be a non-clique sub*slice with respect to some LBFS ordering τ. Then in any LBFS ordering σ of G, x, the last S-vertex in σ has a flying edge xy with respect to S and τ if and only if y is after x in σ. □

Thus for our purposes, a flies if and only if $i_+(a) > \sigma_+(a)$ (similarly b flies if and only if $i_{++}(b) > \sigma_{++}(b)$). To see how steps (c) and (d) are justified we use the following lemma.

Lemma 4.7. Let G be an interval graph and let S be a non-clique sub*slice with respect to a good LBFS ordering τ. Then in any LBFS ordering σ of G, with last S-vertex x, where x does not fly in τ we have the following:
(a) $\mathrm{diam}(S) > 2$:
x neighbour flies if and only if there is a y in S such that $xy \in E$ and $i_\sigma(y) > \sigma(x)$.
(b) $\mathrm{diam}(S) = 2$:
(i) If there is a y in S such that $xy \in E$ and $i_\sigma(y) > \sigma(x)$, then x neighbour flies in S.
(ii) If there is no such y in S, and x neighbour flies, then all neighbours of x that fly are universal to S. □

References

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

[2] D. G. Corneil, S. Olariu, and L. Stewart, The linear structure of graphs: asteroidal triple-free graphs, to appear, *SIAM J. Disc. Math.*, extended abstract appeared in *Proc. 19th International Workshop on Graph Theoretic Concepts in Computer Science, WG'93*, Utrecht, The Netherlands, June 1993, Lecture Notes in Computer Science 790, J. van Leeuwen (Ed.), Springer-Verlag, Berlin, 1994, 211-224.

[3] D. G. Corneil, S. Olariu, L. Stewart, Linear time algorithms for dominating pairs in asteroidal triple-free graphs, to appear, *SIAM J. on Computing*, extended abstract appeared in *Proc. 22nd International Colloquium on Automata, Languages, and Programming ICALP '95*, Lecture Notes in Computer Science 944, Z. Fülöp, F. Gécseg (eds.), Springer-Verlag, Berlin, 1995, 292–302.

[4] P. C. Gilmore and A. J. Hoffman, A characterization of comparability graphs and interval graphs, *Canadian Journal of Mathematics*, **16**, (1964), 539–548.

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

[6] M. Habib, C. Paul and L. Viennot, Lex-BFS, a partition refining technique. Application to transitive orientation, interval graph recognition and consecutive 1's testing, submitted for publication (1997).

[7] W.-L. Hsu, A simple test for interval graphs, *Proc. 19th International Workshop on Graph Theoretic Concepts in Computer Science, WG'92*, Wiesbaden-Naurod, Germany, June 1992, Lecture Notes in Computer Science 657, E. W. Mayr (Ed.), Springer-Verlag, Berlin, 1993, 11-16.

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

[9] C. G. Lekkerkerker and J. C. Boland, Representation of a finite graph by a set of intervals on the real line, *Fundamenta Mathematicae,* **51**, (1962), 45–64.

[10] T. Ma, unpublished manuscript.

[11] S. Olariu, An optimal greedy heuristic to color interval graphs, *Information Processing Letters*, **37**, (1991), 65–80.

[12] K. Simon, A new simple linear algorithm to recognize interval graphs, *Proc. International Workshop on Computational Geometry, CG'91*, Bern, Switzerland, March 1991, Lecture Notes in Computer Science 553, H. Bieri and H. Noltemeyer (Eds.), Springer-Verlag, Berlin, 1992, 289-308.

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