Optimal parallel algorithms for finding cut vertices and bridges of interval graphs

Alan P. Sprague
Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294, USA

K.H. Kulkarni
Rust College, Holly Springs, MS 38635, USA

Communicated by S.G. Akl
Received 14 June 1991
Revised 20 February 1992

Abstract


We present O(log n) time algorithms in the EREW PRAM model, using n/log n processors, to find cut vertices, bridges, and blocks (often called biconnected components) of an interval graph having n vertices. It is assumed the interval graph is represented by an interval model, with ends presorted. If the ends are not presorted, our algorithms, preceded by an optimal sort, form an O(log n) time algorithm using n processors, which is shown to be optimal. The algorithms rely heavily on the parallel prefix algorithm.

Keywords: Parallel algorithms, interval graphs, cut vertices, bridges

1. Introduction

A graph is an interval graph if its vertices can be put in one-to-one correspondence with a family of intervals on the real line in such a way that two vertices are adjacent if and only if the corresponding intervals are nondisjoint. One application of interval graphs is circuit routing [16]; other applications such as archaeological seriation and genetic sequencing are described in [7].

Bertossi and Bonuccelli gave parallel algorithms for several problems on interval graphs [3].

More recently, Kim [9] and Olariu et al. [11] gave optimal parallel algorithms for various problems, including those treated nonoptimally by Bertossi and Bonuccelli; their approach resembled the approach of [3], but their techniques were optimal. These O(log n) time algorithms (in the EREW PRAM model) include algorithms for maximum independent set and minimum dominating set [9,11], center of an interval graph [11], and minimum clique cover and breadth first and depth first search trees [9]. In both [9,11] it is assumed that the interval graph is given by a set of n intervals on the real line. In [9] it is assumed that the ends of intervals are already sorted, and the algorithms contained therein have cost (that is, processor-time product) O(n), which is clearly optimal. In [11] the ends are not assumed to be

Correspondence to: Dr. A.P. Sprague, Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294, USA.
2. Characterization of cut vertices and bridges

Throughout this and the following section $G$ is an interval graph having $n$ vertices. An interval model for $G$ is given: vertices of $G$ are intervals on the real line, and two vertices $x$ and $y$ are adjacent if and only if $x$ and $y$ as intervals are nonadjacent. No two ends have the same coordinate. Further, it is assumed that the ends of the intervals are sorted. Let $LR\_array[1:2n]$ be the $2n$ ends, in ascending order. Each entry $LR\_array[i]$ contains a pointer to the interval whose end is $i$th in sorted order, and an indication of which end (left or right) it is.

By replacing coordinates of ends by their indices in $LR\_array$, we may presume that coordinates of ends are the $2n$ integers from 1 to $2n$: the coordinate of each end equals its index in $LR\_array$. For any interval $x$, $x_L$ is the coordinate of the left end of $x$, and $x_R$ is the coordinate of its right end.

The sequence of density values of an interval graph gives much useful information about the graph. Informally, the density of coordinate $c$ is the number of intervals which contain $c$. Formally, we define the density only at integers: for integer $k$ ($1 \leq k \leq 2n$), we define the density of $G$ at $k$, denoted $d_k$, to be the number of intervals which contain $k + e$, where $0 < e < 1$. Equivalently, $d_k$ is the number of intervals $x$ such that $x_L \leq k < x_R$. The density sequence is the sequence $(d_1, d_2, \ldots, d_{2n})$. Note that $|d_j - d_{j-1}| = 1$ for all $j$. We say that interval $x$ contributes to $d_k$ if $x_L \leq k < x_R$.

We define the sequence $\bar{b} = (b_1, b_2, \ldots, b_s)$ to be a subsequence of sequence $\bar{a} = (a_1, a_2, \ldots, a_s)$ if for some $h$, $b_i = a_i + h$, for all $i$ ($1 \leq i \leq s$). In this situation we will also say that $\bar{b}$ is a $(b_1, b_2, \ldots, b_s)$-subsequence of $\bar{a}$. Thus, according to this definition, a subsequence of $\bar{a}$ is sequence of contiguous entries in $\bar{a}$.

**Proposition 2.1.** An interval $x$ is a cut vertex if and only if there is an integer $j$ such that $x_L \leq j < x_R$ and $(d_{j-1}, d_j, d_{j+1}) = (2, 1, 2)$.

**Proof.** Let $x$ be a vertex and $(d_{j-1}, d_j, d_{j+1}) = (2, 1, 2)$ for some $j$ such that $x_L \leq j < x_R$. Let $Y$
be the set of intervals $v$ in the component containing $x$, such that $v < j$. Let $Z$ be the set of intervals $v$ in the component containing $x$, such that $v > j$. Since $d_{j-1} = 2$ and $d_{j+1} = 2$, $Y$ and $Z$ are nonempty. Also, every interval, except $x$, in the component containing $x$ is in $Y \cup Z$. Each interval $Y$ is disjoint from each interval in $Z$, so there is no edge in $G$ joining $Y$ with $Z$. Then the removal of $x$ from $G$ increases the number of components of $G$, so $x$ is a cut vertex.

Conversely, let $x$ be a vertex, and for each $j$ such that $x < j < x_R$, $(d_{j-1}, d_j, d_{j+1}) = (2, 1, 2)$. Let $u, v$ be adjacent to $x$; to show $x$ is not a cut vertex it is sufficient to demonstrate that there is a path from $u$ to $v$ not containing $x$. If intervals $u$ and $v$ are nonadjoint, there is a path of length 1. Assume that $u$ and $v$ are disjoint; we may assume that $u < v$. Then $x < u < v < x_R$, since both are adjacent to $x$. It is not hard to see that between $u$ and $v$, the density is always greater than 1, since throughout this region the density sequence has no (2, 1, 2)-subsequence, and $|d_i - d_{i-1}| = 1$ for all $i$. Then if $x$ were removed from $G$ the resulting graph would still have a path from $u$ to $v$. Hence $x$ is not a cut vertex.

Proof. Let the edge $uv$ be a bridge. Then intervals $u$ and $v$ are not disjoint. Let $h = \max(u, v)$, and $k = \min(u, v)$; $h < k$, since the intervals are not disjoint. For each $j$ ($h < j < k - 1$), $d_j = 2$. $d_j > 2$ because $u$ and $v$ contribute to $d_j$; if $d_j > 3$, then $u, v$, and a third vertex $w$ would form a triangle in $G$. Since two consecutive density values are never equal, $h + 1 = k$. Then $(d_{h-1}, d_h, d_{h+1}) = (1, 2, 1)$, since the end at coordinate $h$ is a left end (of either $u$ or $v$) and the end at $h + 1$ is a right end.

Conversely, suppose $(d_{j-1}, d_j, d_{j+1}) = (1, 2, 1)$ for some $j$. Let the two intervals contributing to $d_j$ be $u$ and $v$. Then $uv$ is an edge of $G$. We show $uv$ is a bridge. The right end of $u$ or $v$ is at $j + 1$; say $u = j + 1$. Either $u = j$ or $v = j + 1$. If $u = j$, $u = j + 1$, then $u$ is a degree-1 vertex. It is clear that the edge on a degree-1 vertex is a bridge.

Consider the second case, where $v = j, u = j + 1$. If we were to shorten each of these intervals by making $u = j$ and $v = j + 1$, the resulting graph $G'$ would lose exactly one edge (namely $uv$) and since the density for $G'$ at $j$ would be zero, $G'$ would have one more component than $G$. Since removal of edge $uv$ increases the number of components, $uv$ is a bridge.

In Fig. 1, sequence (2, 1, 2) appears three times as a subsequence of the densities. For two of these the cut vertex is interval $A$, and for the third the cut vertex is $D$. The next proposition states that bridges correspond to (1, 2, 1)-subsequences of the density sequence. The only such subsequence is $(d_{15}, d_{16}, d_{17})$; by Proposition 2.2 the only bridge is the edge $(D, I)$.

**Proposition 2.2.** There is a one-to-one correspondence between bridges in $G$ and (1, 2, 1)-subsequence of the density sequence. For each (1, 2, 1)-subsequence $(d_{j-1}, d_j, d_{j+1})$, the corresponding bridge is the pair of intervals contributing to $d_j$.

![Fig. 1. Interval model for interval graph with nine vertices. Sequence of densities $d_i$, and accompanying index $i$ for each.](image)

3. Parallel algorithms

In this section we present parallel algorithms to find the cut vertices, and blocks of $G$. In the EREW PRAM model these algorithms execute in $O(\log n)$ time, using $O(n/\log n)$ processors. These algorithms rely heavily on the parallel prefix algorithm. It is well known that the parallel prefix algorithm computes an arbitrary associative binary operation in $O(\log n)$ time using $n/\log n$ processors in the EREW PRAM model [6, 1 (p. 346)]. We start this section with the algorithm for cut vertices. It is followed by several paragraphs fleshing out details for the various steps.
Parallel algorithm to find cut vertices
1. Compute the density sequence of \( \hat{d} \).
2. For each \( i \) such that \( d_i > 0 \) find the name of an interval contributing to \( d_i \).
3. Obtain a list of cut vertices (with repetitions).
4. Obtain a list of cut vertices, without repetitions.
5. Compress this list.

The first step in the algorithm is an application of prefix sums. For each \( i \), if the \( i \)th end in \( LR \_array \) is a left end, assign \( a_i \) the value 1, while if it is a right end assign it \(-1\). Then \( d_i = \sum_{j < i} a_j \).

The second step is roughly an application of prefix maxima. For each \( i \), if the \( i \)th end in \( LR \_array \) is the left end of an interval, say interval \( x \), let \( e_i \) be the ordered pair \((x, x_k)\), while if it is a right end, let \( e_i \) be \((?,-\infty)\). Then at each left end the value of \( e_i \) is the name of the interval, followed by the coordinate of its right end, while at each right end \( e_i \) is essentially vacant. Define a binary operation of these pairs: \((x, a) \circ(y, b) = (x, a)\) if \( a > b \), and equals \((y, b)\) otherwise. This operation is associative. Define \( f_i = e_1 \circ e_2 \circ \cdots \circ e_i \). Then for each \( i \), \( f_i \) is a pair whose first coordinate is the name of an interval whose left end occurs at or left of \( i \), and among such intervals extends a maximum distance rightward. Hence for each \( i \) such that \( d_i > 0 \), the first coordinate of \( f_i \) is the name of an interval which contributes to \( d_i \).

The third step is trivial, given the processing of the previous steps. For each \( i \) such that \((d_{i-1}, d_i, d_{i+1}) = (2, 1, 2)\), \( f_i \) contains the name of the interval contributing to \( d_i \). This interval is a cut vertex, by Proposition 2.1; we let \( g_i \) be this cut vertex. For all remaining \( i \) let \( g_i \) be vacant. This list \( \vec{g} \) of cut vertices suffers two undesirable traits— it is sparse in that many of its \( 2n \) entries are vacant, and it may contain repetitions. Steps 4 and 5 rectify these two problems.

In the fourth step, parallel prefix can be used to propagate the names of cut vertices rightward, to replace the vacancies. The result is a list of cut vertices with many repetitions but no vacancies, except preceding the first cut vertex. After this each cut vertex appears in consecutive locations of the sequence. Detecting where this sequence changes value is equivalent to producing a list of cut vertices, without repetitions. In the example of Fig. 1, it would be detected that 14 is the last index in which the propagated cut vertex name is \( A \), and 18 is the last index for cut vertex \( D \).

Compressing, or packing, a sparse array is a standard use of parallel prefix [10]. It is accomplished by putting a value of 1 at each nonvacant entry and 0 at each vacancy, and computing prefix sums. The value of the prefix sum at each nonvacant entry indicates where that entry is to be placed in the compressed list. Where the graphs has \( k \) cut vertices, the final result of the algorithm is the integer \( k \), together with an array of length \( k \) which lists the cut vertices.

Parallel algorithm to find bridges
1. Compute the density sequence \( \hat{d} \).
2. For each \( i \) such that \( d_i > 0 \) find the name of an interval contributing to \( d_i \).
3. Obtain a list of bridges.
4. Compress this list.

The first two steps in this algorithm are the same as in the previous algorithm. Processing for the third step is suggested by Proposition 2.2: for each \((1, 2, 1)\)-subsequence \((d_{i-1}, d_i, d_{i+1})\) of the density sequence, we need to find the names of the two intervals contributing to \( d_i \). Suppose the two intervals contributing to \( d_i \) are \( u \) and \( v \), with \( u_k < v_k \). Step 2, when executed as outlined at the previous algorithm, finds \( v \). Since \( d_{i+1} < d_i \), some interval, namely \( u \), has its right end at \( i+1 \), so \( LR \_array[i+1] \) points to \( u \). Step 4 uses parallel prefix yet again.

The algorithm for finding blocks is fairly simple, after some terminology is established. Cut vertices correspond to \((2, 1, 2)\)-subsequences of densities, and \((1, 0, 1)\)-subsequences correspond to places where the interval graph is disconnected. \((2, 1, 2)\)- and \((1, 0, 1)\)-subsequences are called constrictions; we say that the constriction \((d_{i-1}, d_i, d_{i+1})\) is located at \( i + \varepsilon \) \((0 < \varepsilon < 1)\). In addition, we say that there are constrictions at \(-\infty\) and \(\infty\). Let \( a_0, a_1, \ldots, a_i \) be the locations of restrictions, from left to right: \( a_0 = -\infty \), \( a_i = \infty \), and for each remaining \( a_i \), \( a_i - \varepsilon \) is an integer.
strictly between 0 and $2n$. There is a one-to-one correspondence between blocks and adjacent pairs of constrictions. We say that the name of the block located between $a_i$ and $a_{i+1}$ is $i$, and call the interval between $a_i$ and $a_{i+1}$ the domain of this block. For each $i$ ($0 \leq i < r$) the intervals of block $i$ are those intervals which are not disjoint from the interval $[a_i, a_{i+1}]$. Then an interval $x$ which is not a cut vertex $x$ belongs to a single block, namely the one such that $a_i < x_L$ and $x_R < a_{i+1}$. Each cut vertex belongs to more than one block. The algorithm below labels each vertex with the name of the block(s) it belongs to.

Parallel algorithm to find blocks
1. Compute the density sequence $d'$.
2. Label the domain of each block with the name of the block.
3. Label each left and right end with the name of the block whose domain contains the end.

Step 1 is as before. To perform Step 2 we first mark all constrictions. In particular, at each integer $i$ such that there is a constriction at $i - 1 + \varepsilon$, we assign $\text{mark}[i] = 1$, otherwise $\text{mark}[i] = 0$. Perform prefix sums on the marks. This marks the domain of each block with the name of that block. In the example of Fig. 1, for $i \leq 5$ the label is 0, for $6 \leq i \leq 11$ the label is 1, for $12 \leq i \leq 15$ the label is 2, and for $16 \leq i$ the label is 3.

Step 3 results in each vertex receiving two labels (names of blocks), one for its left end and one for its right. For each vertex which is not a cut vertex, the two labels are the same. For each cut vertex, the first label is the name of the leftmost block containing it, and the second is the rightmost containing it; the cut vertex is contained in precisely those blocks between these two labels, inclusive.

4. Lower bounds

In particular, we show that each of these problems has complexity $\Omega(n \log n)$, where ends are not presorted.

In the uniform gap problem a list $S$ of $n$ numbers and a positive number $\varepsilon$ are given; the objective is to determine if all gaps between adjacent values in $S$ after sorting are equal to $\varepsilon$. This problem is known to have complexity $\Omega(n \log n)$ in the bounded degree algebraic decision tree model [2,12]. Uniform gap was reduced to interval graph connectedness in [11]: all gaps between adjacent values in $S$ after sorting are equal to $\varepsilon$ if and only if interval graph $G$ having as vertex set the $n$ closed intervals $\{[s_j, s_j + \varepsilon]: s_j \in S\}$ is connected and $\max(S) - \min(S) = (n - 1)\varepsilon$.

Let a representation for an interval graph $G$ be given, and let $x$ be an interval which is not a vertex of $G$. We call $x$ a long interval if $x_L$ is less than the minimum left end of the intervals of $G$, and $x_R$ is greater than the maximum right end of the intervals in $G$. Interval graph connectedness may be reduced to the problem of finding cut vertices in an interval graph: interval graph $G$ is connected if and only if the interval graph $G'$ obtained by adding one long interval to $G$ has no cut vertices. This same construction reduces interval graph connectedness to the problem of finding the blocks of an interval graph. Thus the problem of finding cut vertices and blocks of an interval graph have complexity $\Omega(n \log n)$, so long as the ends of the intervals are not presorted.

For any $\alpha > 0$, the method of Ben-Or [2] shows that $\Omega(n \log n)$ is the complexity of the problem of deciding if the minimum gap between any two members of a list $S$ of $n$ numbers is at most $\alpha$. This problem may be reduced to the problem of finding bridges in an interval graph as follows. Given list $S$ of $n$ numbers, construct the interval graph $G$ having as vertices the intervals $\{[s_j, s_j + \alpha]: s_j \in S\}$; let $G'$ be the interval graph consisting of the $n$ intervals of $G$, together with one long interval. Then the minimum gap between members of $S$ is greater than $\alpha$ if and only if $G$ is an edgeless graph, which in turn is equivalent to $G'$ having $n$ bridges. Thus, where the ends of the intervals are not presorted, the problem of finding bridges in an interval graph has complexity $\Omega(n \log n)$. 
5. Conclusion

In this paper we have given parallel algorithms for connectivity related problems on interval graphs. Our methods appear to extend easily to the problem of determining if an interval graph is \(k\)-connected for arbitrary \(k\), but not to \(k\)-edge- connectivity [4].

It has been asked how to treat problem instances where coordinates of ends are not distinct. In this discussion we presume intervals are closed, so intervals \(x\) and \(y\) having \(x_R = y_L\) are non-disjoint. If ends are not presorted, it is desirable to sort them in such a way that, among ends sharing the same coordinate, left ends precede right ends. Once ends are so sorted, the possibility that coordinates of ends are not distinct is totally immaterial; it does no harm to treat the index of each end in the sorted array of left and right ends as its coordinate. If ends come presorted but, among ends sharing the same coordinate, left ends do not necessarily precede right ends, this deficiency may be corrected in \(O(\log n)\) time with \((n/\log n)\) processors using parallel prefix.

References


