Optimal Algorithms for Interval Graphs

YUE-LI WANG, KUO-CHING CHIANG AND MING-SHING YU*

Department of Information Management
National Taiwan University of Science and Technology
Taipei, Taiwan 107, R.O.C.
E-mail: ylwang@cs.ntust.edu.tw
* Department of Applied Mathematics
National Chung-Hsing University
Taichung, Taiwan 402, R.O.C.

In this paper, simple optimal algorithms are presented for solving some problems related to interval graphs. These problems are the connected component problem, the spanning tree problem, the eccentricity problem, and the single source all destinations shortest path problem. All of the above four problems can be solved in linear time if the endpoints of the intervals are sorted. Moreover, our algorithms can be parallelized very easily so that the above problems can be solved in $O(\log n)$ time with $O(n/\log n)$ processors using the EREW PRAM model.

**Keywords:** parallel algorithm, spanning tree, connected component, single source all destinations, eccentricity, interval graph, graph theory, EREW PRAM computational model.

1. INTRODUCTION

Let $G = (V, E)$ be a graph, where $V$ and $E$ are the vertex and edge sets, respectively. $G$ is connected if the members of every pair of points are joined by a path. A maximal connected subgraph of $G$ is called a connected component or simply a component of $G$. Thus, a disconnected graph has at least two components. The connected component problem is to find all the connected components of $G$.

Every connected graph $G$ contains a spanning subgraph that is a tree, called a spanning tree [3]. Typically, there are many different spanning trees in a connected graph. For a spanning tree, there are some properties which are described as follows:

The following are equivalent on a graph $T = (V, E)$, where $n$ is the number of vertices and $m$ is the number of edges.

1. The graph $T$ is a tree.
2. The graph $T$ is connected and $m = n - 1$.
3. Every pair of distinct vertices of $T$ are jointed by a unique path.
4. The graph $T$ is acyclic and $m = n - 1$.

Received October 19, 1995; revised February 27, 1998.
Communicated by Wen-Lian Hsu.
For an undirected unweighted graph, the problem of constructing a spanning tree can be solved in $O(m + n)$ time using the BFS (Breadth-First Search) algorithm [3]. In the parallel case, the problem can be solved in $O(\log n)$ time with $O(m + n)$ processors with the CRCW PRAM (Concurrent-Read-Concurrent-Write Parallel Random Access Machine) computational model using the algorithm to eliminate cycles [9].

In a connected graph $G = (V, E)$, the distance $d_G(u, v)$ between two of its vertices $u$ and $v$ is the length of the shortest path (i.e., the number of edges in the shortest path) between them. Given a source vertex $s$, the single source all destinations problem is to determine the shortest paths from $s$ to all the remaining vertices of $G$[1, 4, 5]. The eccentricity $e(v)$ of a vertex $v$ is the distance from $v$ to the vertex farthest from $v$. The diameter and radius of a connected graph are the largest and the smallest, respectively, eccentricities in the graph [7]. In general, computing the eccentricities of all of the vertices in a connected graph takes $O(n^3)$ time. In [13], Wang and Chang gave an $O(m + n)$ algorithm to find the eccentricities of all of the vertices in a connected interval graph. Thus, the diameter and the radius of a connected interval graph can also be computed in $O(m + n)$ time.

An interval family $I$ is a set of intervals on a real line. An undirected graph $G = (V, E)$ is called an interval graph if its vertices can be put into a one-to-one correspondence with an interval family $I$ such that two vertices are adjacent in $G$ if and only if their corresponding intervals have nonempty intersection. The interval family $I$ is called an interval model of $G$ [6]. A wide variety of researches on interval graphs have been carried out [11, 12, 14, 15].

Let $G$ be the interval graph associated with the interval model $I_G = \{1, \ldots, n\}$, where interval $i$ is represented by $[a_i, b_i]$ for $i = 1, \ldots, n$ with $a_i$ being the left endpoint of interval $i$ and $b_i$ being its right endpoint. For simplicity, we assume that these $2n$ endpoints are distinct. Thus, we can label the intervals in increasing order of their left endpoints. For example, Fig. 1(a) is an interval graph $G$, and Fig. 1(b) is its corresponding interval model $I_G$ with labelled intervals.

![Diagram](image-url)
In this paper, we will consider some problems for an unweighted interval graph $G$. We shall propose linear time algorithms for solving the connected component problem, the spanning tree problem, the eccentricity problem, and the single source all destinations problem if the endpoints of the intervals are sorted. Our algorithms can be parallelized so that the above problems can be solved in $O(\log n)$ time with $O(n/\log n)$ processors using the EREW PRAM (Exclusive-Read-Exclusive-Write Parallel Random Access Machine) computational model.

The remainder of this paper is organized as follows. In Section 2, we propose a sequential algorithm which can be parallelized to find the connected components of an interval graph. In Section 3, we introduce a sequential algorithm which can also be parallelized to construct a spanning tree of an unweighted connected interval graph. In Section 4, we show how we can compute the eccentricities of all the vertices in a connected interval graph using the previous results. In Section 5, we give an algorithm to solve the single source all destinations problem. Finally, concluding remarks are presented in Section 6.

2. AN OPTIMAL ALGORITHM FOR FINDING CONNECTED COMPONENTS

In this section, we will propose an algorithm for finding the connected components of an interval graph. Assume that the endpoints of the intervals are sorted in increasing order. Let $S = s_1 s_2 \ldots s_{2n}$ denote the sorted endpoint sequence, where $n$ is the number of the intervals. For example, the following sequence is, in increasing order, the endpoints of the interval model $I_G$ in Fig. 1:

$$S = a_1a_2a_3b_2a_4b_1b_3a_5a_6b_4a_7a_8b_6b_5a_9b_7a_{10}b_9b_8b_{10}.$$

That is, $s_1 = a_1$, $s_2 = a_2$, etc. For each $s_i$, $i = 1$, $2$, $\ldots$, $2n$, we define

$$l_i = \begin{cases} j & \text{if } s_i = a_j \\ \infty & \text{otherwise} \end{cases},$$

and

$$r_i = \begin{cases} j & \text{if } s_i = b_j \\ \infty & \text{otherwise} \end{cases}.$$

and

$$x_i = \min\{r_i, r_{i+1}, \ldots, r_{2n}\}.$$

For example, Fig. 2 (a) is a disconnected interval graph $G$. Fig. 2 (b) show its corresponding interval model $I_G$ with labelled intervals. Fig. 2 (c) is the $r_i$, $l_i$, $x_i$ of the model $I_G$.

Algorithm A, described as follows, is used to detect the connected components of an interval graph.
Algorithm A

Input: An interval model of $G$ and its corresponding sorted endpoints sequence $S = s_1 s_2 \ldots s_{2n}$.

Output: The connected components of $G$.

Method:

Step 1: Compute $r_i$ and $l_i$ for $i = 1, 2, \ldots, 2n$.

Step 2: Compute $x_i$ for $i = 1, 2, \ldots, 2n$.

Step 3: Find all $l_i$'s such that $l_i = x_i$. Let $\{j_1, j_2, \ldots, j_k\}$ be the set which contains all the found $l_i$'s and $j_1 < j_2 < \ldots < j_k$.

Step 4: The connected components of $G$ are $\{j_1, j_1+1, \ldots, j_2-1\}, \{j_2, j_2+1, \ldots, j_3-1\}, \ldots, \{j_k, j_k+1, \ldots, n\}$.

End of Algorithm A

For an example, see Fig. 2 again. $[1, 3, 6]$ is the set which contains all the found $l_i$'s in Step 3 of Algorithm A. We obtain that $G$ has three components: $[1, 2], [3, 4, 5], [6, 7, 8, 9]$.

Step 1 of Algorithm A takes $O(n)$ time. Since $x_i = \min[r_i, x_{i-1}], i = 1, 2, \ldots, 2n - 1$, and $x_{2n} = r_{2n}$, Step 2 can also be done in $O(n)$ time. Clearly, Steps 3 and 4 also take $O(n)$ time. Therefore, Algorithm A uses $O(n)$ time to find the connected components of an interval graph. If the endpoints are not sorted, then our algorithm has to take $O(n \log n)$ time.
Algorithm A can be parallelized on the EREW PRAM model very easily. Obviously, Steps 1, 3 and 4 can be computed in O(log n) time using O(n/log n) processors. Applying the parallel prefix technique [10], Step 2 can be done in O(log n) time with O(n/log n) processors. Thus, Algorithm A will take O(log n) time with O(n/log n) processors using the EREW PRAM model if the endpoints are sorted. Otherwise, our algorithm takes O(log n) time with O(n) processors since we have to sort 2n endpoints [2].

Now, we will prove the correctness of Algorithm A. Let G be an interval graph, let \( S = s_1 \ s_2 \ldots \ s_{2n} \) be the sorted sequence of the endpoints and let \( \{ j_1, j_2, \ldots, j_k \} \) be the set which contains all the found \( l_i \)'s in Step 3 of Algorithm A with \( j_1 < j_2 < \ldots < j_k \).

**Theorem 1:** G is connected if and only if \( l_i \neq x_i \) for \( 1 < i \neq 2n \).

**Proof:** Following from the definition of \( x_i \), intuitively, \( x_i \) is the interval with the smallest left endpoint in the set of intervals whose right endpoints are not less than endpoint \( s_i \). Thus, if \( l_i \neq \infty \) and \( l_i \neq x_i \), then there exists an interval which contains left endpoint \( s_i \). If \( l_i = x_i \), then there does not exist any interval which contains left endpoint \( s_i \). It is easy to verify that G is connected if and only if \( l_i \neq x_i \) for \( 1 < i \neq 2n \). Q.E.D.

**Corollary 1:** \( \{ j_1, j_1 + 1, \ldots, j_2 - 1 \}, \{ j_2, j_2 + 1, \ldots, j_3 - 1 \}, \ldots, \{ j_k, j_k + 1, \ldots, n \} \) are the connected components of G.

### 3. AN OPTIMAL ALGORITHM FOR CONSTRUCTING A SPANNING TREE

In this section, we will introduce an optimal algorithm, which is described as Algorithm B, for constructing a spanning tree of an unweighted connected interval graph. In the remainder of this paper, we will assume that the interval graph G is connected.

**Algorithm B**

**Input:** An interval model of an unweighted connected interval graph G. Assume that the 2n endpoints are sorted in increasing order. Let \( S = s_1 \ s_2 \ldots \ s_{2n} \) be the sorted endpoint sequence.

**Output:** A spanning tree T of G.

**Method:**

**Step 1:** Initially, let \( T = (V, E) \) be a graph, where \( V = \{ 1, 2, \ldots, n \} \) and \( E = \emptyset \).

**Step 2:** Compute \( r_i \) and \( l_i \) for \( i = 1, 2, \ldots, 2n \).

**Step 3:** Compute \( x_i \) for \( i = 1, 2, \ldots, 2n \).

**Step 4:** For \( i = 2 \) to \( 2n \), if \( l_i \neq \infty \), then \( E = E \cup (l_i, x_i) \). The resulting T is a spanning tree of G.

**End of Algorithm B**
For an example, see the graph in Fig. 1 again. After Algorithm B is terminated, we obtain a spanning tree $T$ which contains nine edges (2, 1), (3, 1), (4, 1), (5, 4), (6, 4), (7, 5), (8, 5), (9, 7), and (10, 8) (see Fig. 3 (a)). We show the spanning tree in Fig. 3 (b).

\[
\begin{array}{cccccccccccccccc}
 i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\
 s_i & a_1 & a_2 & a_3 & b_2 & a_4 & b_1 & b_3 & a_5 & a_6 & b_4 & a_7 & a_8 & b_6 & b_5 & a_9 & b_7 & a_{10} & b_9 & b_8 & b_{10} \\
 r_i & \infty & \infty & \infty & 2 & \infty & 1 & 3 & \infty & \infty & 4 & \infty & \infty & 6 & 5 & \infty & 7 & \infty & 9 & 8 & 10 \\
 l_i & 1 & 2 & 3 & \infty & 4 & \infty & \infty & 5 & 6 & \infty & 7 & 8 & \infty & \infty & 9 & \infty & 10 & \infty & \infty & \infty \\
x_i & 1 & 1 & 1 & 1 & 3 & 4 & 4 & 4 & 5 & 5 & 5 & 7 & 7 & 8 & 8 & 8 & 8 & 10 \\
\end{array}
\]

(a)

(b)

Fig. 3. (a) Illustration of Algorithm B. (b) The obtained spanning tree.

Obviously, Steps 1 and 4 of Algorithm B take $O(n)$ time. Steps 2 and 3 take $O(n)$ time as described in the analysis of Algorithm A. Therefore, a spanning tree of an interval graph can be constructed in linear time using Algorithm B. Algorithm B can also be parallelized on the EREW PRAM model and takes $O(\log n)$ time using $O(n/\log n)$ processors. Notice that if the endpoints are not sorted, then our algorithm will take $O(n\log n)$ time in sequential and take $O(\log n)$ time with $O(n)$ processors in parallel using the EREW PRAM model.

Now, we will prove the correctness of Algorithm B.

**Theorem 2:** Let $G$ be a connected interval graph and $T = (V, E)$ be the graph found by Algorithm B. Then, $T$ is a spanning tree of $G$.

**Proof:** Every vertex which is not equal to 1 in $V$ is adjacent to a smaller vertex. Hence, there are $n - 1$ edges which will be selected by Algorithm B. If we can prove that there is no cycle in $T$, then $T$ is a spanning tree of $G$. For the purpose of contradiction, we assume that there is a cycle $v_1, v_2, \ldots, v_p, v_1$ in $T$. Assume without loss of generality that $v_1 < v_2$. Let $\text{Adj}(u)$ denote the adjacency set of vertex $u$ in $T$. For every vertex $u \in V$ and $u \neq 1$, there is exactly one vertex $v \in \text{Adj}(u)$ which is less than $v$, $v_1 < v_2 < \cdots < v_p < v_1$. This is a contradiction. This completes the proof. Q.E.D.
4. AN OPTIMAL ALGORITHM FOR COMPUTING ECCENTRICITIES

In this section, we will show how we can compute the eccentricities of all of the vertices in a connected interval graph $G$. To compute the eccentricities, we need the following Lemma. Note that $d_G(u, v)$ denotes the length of the shortest path from $u$ to $v$ in a graph $G$.

**Lemma 1:** Let $G = (V, E)$ be a connected interval graph. For any vertex $v \in V$, $d_G(1, v) = d_T(1, v)$, where $T$ is the tree found by Algorithm B.

**Proof:** Let $T$ be the resulting spanning tree of Algorithm B, let $(v, u)$ be an edge in $T$, and let $Adj(v)$ be the adjacency set of vertex $v$ in $G$. Assume without loss of generality that $v > u$. Since $u = \min\{x \mid x \in Adj(v)\}$, the left endpoint of $u$ has the smallest coordinate among the left endpoints of the intervals in $Adj(v)$. This means that $v$ walks to the left as far as possible in one step. Therefore, $d_T(1, v)$ must be equal to $d_G(1, v)$. 

Q.E.D.

**Algorithm C**

**Input:** An interval graph $G = (V, E)$ and its corresponding model, where $V = [1, 2, ..., n]$ and the endpoints are sorted.

**Output:** The eccentricities of all vertices in $G$.

**Method:**

**Step 1:** Find the interval having the leftmost right endpoint and the interval having the rightmost left endpoint. Let $l$ and $r$, respectively, be these two intervals.

**Step 2:** Extend $l$ (respectively, $r$) so that the left (respectively, right) endpoint of $l$ (respectively, $r$) becomes the leftmost (respectively, rightmost) endpoint (see Fig. 4).

```
          ______________
         |            |
         |            |

         l            r
```

Fig. 4. Extend intervals of an interval model.

**Step 3:** Use Algorithm B to find a spanning tree $T$ on this modified model and compute $d_G(1, v)$ from $T$ for all $v \in V$. Using the same technique but scanning from right to left, we can compute $d_G(n, v)$ for all $v \in V$.

**Step 4:** For all $v \in V$, $e(v) = \max\{d_G(1, v), d_G(n, v)\}$.

**End of Algorithm C**

Assume that the endpoints of an interval model are sorted. Step 1 of Algorithm C takes $O(n)$ time to find the two intervals $l$ and $r$. Step 2 can be done in linear time to relabel the intervals according to their left (respectively, right) endpoints. Algorithm B uses $O(n)$ time to find a spanning tree. We can use the BFS algorithm
to compute \( d_G(1, v) \) (respectively, \( d_G(n, v) \)) for all \( v \in V \) after the tree is found. Thus, Step 3 takes \( O(n) \) time. Clearly, Step 4 also takes \( O(n) \) to compute \( e(v) \) for all \( v \in V \). Therefore, the time complexity of Algorithm C is \( O(n) \). Algorithm C can also be parallelized so that it is a parallel algorithm. Steps 1, 2 and 4 can be done in \( O(\log n) \) time using \( O(n/\log n) \) processors on the EREW PRAM model. Applying parallel list ranking and the Euler tour technique [8], Step 3 can also be done in \( O(\log n) \) with \( O(n/\log n) \) processors on the EREW PRAM model.

Now, we will show the validity of Algorithm C.

**Theorem 3.** \( e(v), v \in V \), found by Step 4 of Algorithm C is the eccentricity of \( v \) in \( G \).

**Proof:** Let \( I_G \) be an interval model of \( G \), and let \( I'_G \) be the extended interval model of \( I_G \) after Step 2 of Algorithm C. The interval which has the leftmost right (respectively, rightmost left) endpoint will be labelled 1 (respectively, n) after it is extended. By Lemma 1, \( d_G(1, v) \) and \( d_G(n, v) \) can be computed in Step 3 of Algorithm C.

All we need to prove is that for any other vertex \( u \) in \( G \), \( d_G(u, v) \leq \max \{ d_G(1, v), d_G(n, v) \} \). We will first prove the case where \( a_u \) is on the left hand side of \( a_v \). Since the intervals of \( I'_G \) are labelled in increasing order of their left endpoints, \( d_G(u, v) \) must be less than or equal to \( d_G(1, v) \). Therefore, \( d_G(u, v) \leq d_G(1, v) \leq \max \{ d_G(1, v), d_G(n, v) \} \). Similarly, \( d_G(u, v) \leq d_G(n, v) \leq \max \{ d_G(1, v), d_G(n, v) \} \) if \( a_u \) is on the right hand side of \( a_v \).

Q.E.D.

5. AN OPTIMAL ALGORITHM FOR SOLVING THE SINGLE SOURCE ALL DESTINATIONS PROBLEM

In this section, we will give an optimal algorithm for solving the single source all destinations problem in a connected interval graph \( G \). Algorithm D, which is used to solve the problem, is described below.

**Algorithm D**

**Input:** An interval model \( I_G \) of a connected interval graph \( G = (V, E) \) whose endpoints are sorted and a source vertex \( s \).

**Output:** The lengths of the shortest paths from \( s \) to all the remaining vertices of \( G \).

**Method:**

**Step 1:** Remove all intervals which are less than \( s \) and do not intersect \( s \).

**Step 2:** Extend \( s \) so that its left endpoint becomes the leftmost endpoint on the remaining intervals.

**Step 3:** Use Algorithm B to find a spanning tree and compute \( d_G(s, v) \) from this tree for all \( v \in V \) and \( v > s \).

**Step 4:** Using the same technique but scanning from right to left, we can compute \( d_G(s, v) \) for all \( v \in V \) and \( v < s \).

**End of Algorithm D**
Assume that the endpoints of $I_G$ are sorted. Step 1 of Algorithm D takes $O(n)$ time in sequential and $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model in parallel. Steps 2 and 3 of Algorithm D take $O(n)$ time since they are similar to Steps 2 and 3 of Algorithm C. Step 4 also takes $O(n)$ time. Thus, the time-complexities of Algorithm D is $O(n)$ time in sequential and $O(\log n)$ time with $O(n/\log n)$ processors in parallel on the EREW PRAM model.

Now, we will show the correctness of Algorithm D.

**Theorem 4.** Let $G = (V, E)$ be a connected interval graph and $s$ be a given vertex. Algorithm D computes the lengths of the shortest paths from $s$ to all the remaining vertices of $G$.

**Proof:** We need only prove that Algorithm D can compute $d_G(s, v)$ for all $v \in V$ and $u > s$. Using similar reasoning, we can prove that Algorithm D can also compute $d_G(s, v)$ for all $v \in V$ and $v < s$.

Clearly, removing the intervals from Step 1 of Algorithm D will not change the value of $d_G(s, v)$ for all $v \in V$ and $v > s$. Let $G'$ and $G''$ be the subgraphs of $G$ after Steps 1 and 2, respectively, of Algorithm D. Clearly, $G'$ and $G''$ are the same graph. Therefore, by Lemma 1, Step 3 of Algorithm D can compute $d_G(s, v)$ for all $v \in V$ and $v > s$.

### 6. CONCLUDING REMARKS

In this paper, we have presented four simple optimal algorithms to solve the connected component problem, the spanning tree problem, the eccentricity problem, and the single source all destinations problem for an interval graph. All of them can be done in linear time if the endpoints of the intervals are sorted and can be parallelized very easily.

For the all pairs shortest distances problem, we can let each vertex be the source and repeat Algorithm D. Thus, the all pairs shortest distances problem on interval graphs can be solved in quadratic time and can be parallelized on the EREW PRAM model.

We are currently studying these problems for circular-arc graphs and other special graphs using a similar technique.

### REFERENCES


**Yue-Li Wang** (王有禮) received the B.S. and M.S. degrees from Tamkang University, Taiwan, the Republic of China, in 1975 and 1979, respectively, and the Ph. D. degree from National Tsing Hua University, Taiwan, in 1988, all in computer science. He joined the National Taiwan Institute of Technology in 1990. He is presently the head of the Department of Information Management, National Taiwan University of Science and Technology (originally, the National Taiwan Institute of Technology). Before joining National Taiwan University of Science and Technology, he worked at the Advanced Technology Center, Electronics Research and Service Organization, Industrial Technology Research Institute. His research interests include graph theory, computational geometry, and the design and analysis of computer algorithms.
Kuo-Ching Chiang（蔣國經）was born on August 18, 1970. He received the B.S. and M.S. degrees from the Department of Information Management, National Taiwan Institute of Technology, in 1992 and 1994, respectively. Currently, he works for the Institute for Information Industry. His research interests include computer algorithms and graph theory.

Ming-Shing Yu（余明興）received the B.S., M.S., and Ph. D. degrees in Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, in 1973, 1975, and 1979, respectively. He is currently an associate professor in the Department of Applied Mathematics, National Chung-Hsing University, Taichung, Taiwan. Dr. Yu is a member of both IEEE and ACM. His research interests include speech processing and parallel computing.