Some Optimal Parallel Algorithms for Shortest Path Related Problems on Interval and Circular-arc Graphs

F.R. Hsu * M.K. Shan † H.S. Chao ‡

R.C.T. Lee §

Providence University, Shalu, Taichung Hsien, Taiwan 433, Republic of China

Abstract

In this paper, we consider some shortest path related problems on interval and circular-arc graphs. For the all-pair shortest path query problem on interval and circular-arc graphs, instead of using the sophisticated technique, we propose simple parallel algorithms using only the parallel prefix and suffix computations and the Euler tour technique. Our preprocessing algorithms run in $O(\log n)$ time using $O(n/\log n)$ processors. Using the data structure constructed by our preprocessing algorithms, a query of the length of a shortest path between any two vertices can be answered in constant time by using a single processor.

For the hinge vertex problem on interval graphs, we propose an $O(\log n)$ time algorithm using $O(n/\log n)$ processors. It leads to a linear time sequential algorithm. Our algorithms are on the EREW PRAM model.

1 Introduction

A graph $G = (V, E)$ is an *interval graph* (*circular-arc graph*) if its vertices can be put in a one-to-one corresponded with a set F (S) of intervals (circular-arc) on a real line (unit circle) such that two vertices are adjacent in G if and only if their corresponding intervals (circular-arc) have nonempty intersection. Such a set F (S) is called an *interval* (*circular-arc*) model of the interval (circular-arc) graph G. Interval graphs and circular-arc graphs arise in many application areas, such as scheduling, traffic control, biology, and VLSI design. There is an extensive discussion on these graphs in [8].

Let $G = (V, E)$ be an undirected, unweighted, connected graph and let $|V| = n$ and $|E| = m$. A path between two vertices s and t is a sequence of vertices $(v_1, v_2, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i < k$, where $s = v_1$ and $t = v_k$. The length of the path is the number of edges in the sequence. A shortest path between two given vertices is a path with the shortest length. The *all-pair shortest path problem* is the problem of finding the shortest paths between all pairs of vertices. In this paper, we consider the distance version of the all-pair shortest path problem; i.e. the problem of finding the lengths of the shortest paths between all pairs of vertices.

Given an undirected graph G of n vertices, computing the lengths of the shortest paths between all pairs of vertices in G takes $\Omega(n^2)$ time, since there are $\Theta(n^2)$ pairs of vertices. For general graphs, the best known algorithm was proposed by Seidel [13] and runs in $O(M(n)\log n)$ time, where $M(n)$ is the time necessary to multiply two $n \times n$ matrices of small integers, which is currently $O(n^{2.376})$ [6].

In stead of finding all pairs of shortest paths, a different and much more efficient approach is described as follows: First, apply a faster preprocessing algorithm and construct a data structure. Using the data structure, a query on the length of a shortest path between any two vertices can be answered very fast (e.g., $O(1)$ time). We call such problem the *all-pair shortest path query problem*.

In [12], Olariu et al. proposed an $O(\log n)$ time algorithm using $O(n)$ processors to compute a shortest path in an interval graph on the EREW PRAM. It takes $O(n \log n)$ operations to compute a *single* shortest path. There was no query version

*Corresponding Author, Computer and Communication Center, Providence University, Taiwan, ROC, e-mail:frhsu@pu.edu.tw

†Department of Computer Science, National Chengchi University, Taiwan, R.O.C.

‡Department of Computer Science, National Tsing Hua University, Taiwan, ROC.

§Department of Computer Science and Information Engineering National Chi-Nan University Taiwan, ROC.

proposed in [12]. For the interval and circular-arc graphs, Chen and Lee [5] gave a parallel preprocessing algorithm which runs in $O(\log n)$ time using $O(n/\log n)$ processors on the CREW PRAM. Using the data structure constructed by their preprocessing algorithm, a query of the length of a shortest path between any two vertices can be answered in $O(1)$ time by using a single processor.

In [5], Chen and Lee made use of the technique of the level-ancestor query in trees introduced by Berkman and Vishkin [2]. The technique of level-ancestor query in trees is interesting and useful in many problem contexts [5, 4]. However, it is a fairly hard implemented algorithm.

In this paper, we consider some shortest path related problems on interval and circular-arc graphs. We construct some useful data structures for interval graphs in advance. These data structures are used to solve some problems on interval graphs. In this paper, we first consider the all-pair shortest path query problem on interval and circular-arc graphs. Instead of using the sophisticated technique of level-ancestor queries in trees [2], we propose simple parallel algorithms using only the parallel prefix and suffix computations [1] and the Euler tour technique [1, 14]. Our preprocessing algorithms run in $O(\log n)$ time by using $O(n/\log n)$ processors. Using the data structure constructed by our preprocessing algorithms, a query of the length of a shortest path between any two vertices can be answered in $O(1)$ time by using a single processor. Our algorithms are on the EREW PRAM rather than the CREW PRAM and easy to implement.

In this paper, we also consider the problem for finding all hinge vertices on interval graphs. We call a vertex u is a hinge vertex, if there exist any two vertices in G whose length becomes longer when u is removed. A graph without hinge vertices is called a *self-repairing graph*. For the design of communication network, it is important to analyze topological properties. In a network, the overall communication cost is increased if a computer corresponding to a hinge vertex stalls. Hence finding hinge vertices in a graph is useful for finding critical nodes in an actual network. There exists an $O(n^3)$ sequential algorithm for finding all hinge vertices of a general graph [11]. In [9], Honma and Masuyama proposed an $O(\log n)$ time parallel algorithm using $O(n)$ processors on the CREW PRAM for finding all hinge vertices of an interval graph. It takes $O(n \log n)$ work. In this paper, we propose an $O(\log n)$ time parallel algorithm using $O(n/\log n)$ processors on the EREW PRAM for finding all hinge vertices on interval graphs. Our algorithm results in a linear time sequential algorithm for finding all hinge vertices on interval graphs.

The rest of this paper is organized as follows. Section 2 describes basic notations and some interesting properties and data structures of interval graphs. Section 3 gives algorithms for the all-pair shortest path query problem on interval graphs. Section 4 gives algorithms for the all-pair shortest path query problem on circular-arc graphs. Section 5 shows the algorithm for finding all hinge vertices on interval graphs. Finally, we conclude our results in Section 6.

2 Preliminaries

In this section, we propose some useful data structures and interesting properties on interval graphs which will be used in our algorithms. Assume that the interval graph G is given by its interval model $F = \{I_1, I_2, \ldots, I_n\}$ in sorted order. Let every interval $I_i \in F$ be defined by its left endpoint a_i and right endpoint b_i with $a_i \leq b_i$, for every $i, 1 \leq i \leq n$. Let M be the sorted array containing the endpoints of the intervals in F. Let $M = (p_1, p_2, \ldots, p_{2n})$, where every p_i is either the left or right endpoint for some I_j. Note that from M it is not difficult to obtain the list of all intervals in F that are sorted by the $a_i's$ (respectively, $b_i's$). Hence, we can label intervals in F such a way that $b_i < b_j$ if and only if $i < j$. Such labelling can be easily obtained from the sorted array of F in $O(\log n)$ time using $n/\log n$ processors by doing parallel prefix computation [1]. Since endpoints are sorted, we can replace the real value of an endpoint by its rank in the sorted order. Therefore, we can assume all endpoints are distinct with coordinates of consecutive integer values $1, 2, \ldots, 2n$. We also assume that the input interval graph is connected. It is easy to see that our algorithms can be modified to handle the cases when the input graph is not connected.

In [5], Chen et al. defined a successor function on intervals. In [3], Bespamyatnikh et al. defined a right successor and a left successor of an endpoint. Similar to their idea, we define a leftmost neighbor and a rightmost neighbor of an interval. For each interval I_i, among intervals intersect I_i, consider intervals with rightmost and leftmost endpoints respectively. Formally, let $RMOST(i) = \max\{j | I_j \text{ contains } b_i\}$ and $LMOST(i) = k$ where a_k is equal to $\min\{a_j | I_j \text{ contains } a_i\}$. For ex-

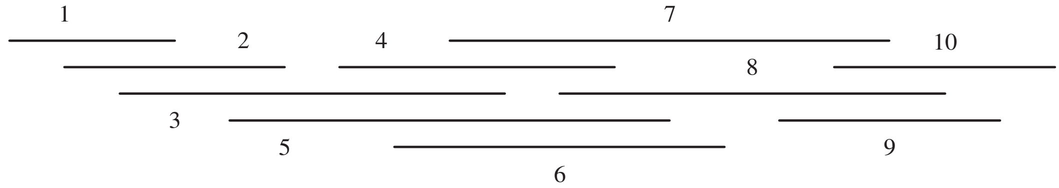

Figure 1: A set of intervals.

ample, in Figure 1, the array $RMOST[1,\ldots,n]$ is equal to $(3,5,7,8,8,8,10,10,10,10)$. Besides, the array $LMOST[1,\ldots,n]$ is equal to $(1,1,1,3,2,3,3,5,7,7)$. Suppose p is an endpoint of interval I_i. For ease of reference, let $A(p)$ and $B(p)$ denote a_i and b_i respectively. Due to the following lemma, array $RMOST[1,\ldots,n]$ and $LMOST[1,\ldots,n]$ are easily obtained in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM by utilizing parallel prefix and suffix computations [1].

Lemma 1 *For any interval I_i, the following statements hold:*

1. $RMOST(i) = k$ if and only if $b_k = \max\{B(s)|s = 1,2,\ldots,b_i\}$,
2. $LMOST(i) = k$ if and only if $a_k = \min\{A(s)|s = a_i, a_i+1,\ldots,2n\}$.

Proof. We prove the first statement only. The proof of the other statement is similar and omitted. Suppose $RMOST(i) = k$. By definition, $RMOST(i) = \max\{j|I_j \text{ contains } b_i\}$. It follows $a_k < b_i$. Let J denote the set of intervals whose any endpoint lies left of b_i. That is $J = \{I_j|a_j < b_i\}$. It follows $I_k \in J$. Furthermore, J can be partitioned into two sets $\{I_j|I_j \text{ contains } b_i\}$ and $\{I_j|b_j < b_i\}$. Consider the set $\{I_j|b_j < b_i\}$. Note that for any interval I_t in $\{I_j|b_j < b_i\}$, we have $b_t < b_i \leq b_k$. It follows that intervals in $\{I_j|b_j < b_i\}$ have no contribution to $RMOST(i)$. Therefore, we can compute $RMOST(i)$ by finding the maximum right endpoint among all intervals in J. □

Besides, for each interval I_i, among intervals intersect I_i, let $RSEC(i)$ denote the interval with the second largest right endpoint. Let $RSEC(i) = i$ if no such interval exists. Let $\text{smax}\{s_1,s_2,s_3,\ldots,\}$ denote the second largest element in set $\{s_1,s_2,s_3,\ldots\}$. Formally, let $RSEC(i) = \max\{i, \text{smax}\{j|I_j \text{ contains } b_i\}\}$. Consider Figure 1 again. The array $RSEC[1,\ldots,n]$ is equal to $(2,3,6,7,7,7,9,9,9,10)$. Similar to Lemma 1, we have the following lemma.

Lemma 2 *For any interval I_i, $RSEC(i) = k$ if and only if $b_k = \max\{i, \text{smax}\{B(s)|\ s = 1,2,\ldots,b_i\}\}$.* □

Now, we show how to compute $\text{smax}\{B(s)|s = 1,2,\ldots,b_i\}$ for $i = 1,n$. Suppose that R is a set of numbers. We define a new operation max2 such that $\text{max2}(R)$ is equal to $\{\max(R), \text{smax}(R)\}$. It is obvious that max2 is an associative operation. Since the max2 operation is associative, we can apply the parallel prefix computations [1] to compute $\text{max2}\{B(s)|s = 1,2,\ldots,b_i\}$ for $i = 1,n$, in $O(\log n)$ time using $O(n/logn)$ processors on the EREW PRAM. Since $\text{smax}(R) = \text{smax}(\text{max2}(R))$, by Lemma 2, we can compute array $RSEC[1,\ldots,n]$ with the same time and processor complexity.

Note that $i < j$ if and only if $b_i < b_j$. We have the following lemma:

Lemma 3 *For any two intervals I_i and I_j, if $i < j$, then $RMOST(i) \leq RMOST(j)$ and $RSEC(i) \leq RSEC(j)$.*

Proof. Suppose $i < j$. It follows $b_i < b_j$. Assume that $RMOST(i) > RMOST(j)$. It follows $b_{RMOST(j)} < b_{RMOST(i)}$. By definition, $a_{RMOST(i)} < b_i$ and $b_j \leq b_{RMOST(j)}$. Therefore, $a_{RMOST(i)} < b_i < b_j \leq b_{RMOST(j)} < b_{RMOST(i)}$. We have $a_{RMOST(i)} < b_j < b_{RMOST(i)}$. It follows I_j intersects $I_{RMOST(i)}$. Besides, $RMOST(i) > RMOST(j)$. This contradicts the definition of $RMOST(j)$. Therefore, $RMOST(i) \leq RMOST(j)$. Similarly, we can prove that $RSEC(i) \leq RSEC(j)$. □

According to the $RMOST$ array, we define a tree T_{RMOST} as follows: each interval I_i corresponds a node i in T_{RMOST} and its parent is $RMOST(i)$. For node i and its sibling j, i is on the left side of j if and only if $i < j$. By Lemma 3, a tree T_{RMOST} is well defined by the $RMOST$ array. Consider Figure 1 again. Its corresponding T_{RMOST} is shown in Figure 2. Let $PreOrder(i)$ and $L(i)$ denote the preorder number and the level of interval i in tree

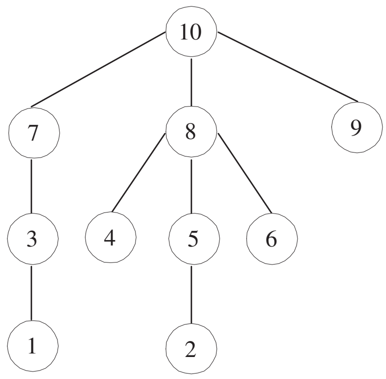

Figure 2: The tree T_{RMOST}.

T_{RMOST}. In this example, the pre-order traversal of T_{RMOST} would be $(10, 7, 3, 1, 8, 4, 5, 2, 6, 9)$ and $PreOrder = (4, 8, 3, 6, 7, 9, 2, 5, 10, 1)$ and $L = (3, 3, 2, 2, 2, 2, 1, 1, 1, 0)$.

Now, consider how to compute array $PreOrder$ and array L. First, each non-root node in T_{RMOST}, we try to find the rank among its siblings. The rank of node i among its siblings is equal to one plus its index i minus the index of its leftmost sibling. By Lemma 3, we observe that the children of each internal node in T_{RMOST} occupy consecutive ranges. For example, in Figure 2, the children of node 8 are from node 4 to node 6. Hence we can decide whether a given node is the leftmost child of its parent in constant time using a single processor. In short, we have the following lemma.

Lemma 4 *For any non-root vertex i in T_{RMOST}, $i > 1$, i is the leftmost child of its parent if and only if $RMOST(i-1) < RMOST(i)$.*

Proof. First, we will prove that the children of each internal node in T_{RMOST} occupy consecutive ranges. Suppose that the leftmost and rightmost child of vertex j's are s and t respectively. Therefore, $RMOST(s) = RMOST(t) = j$. If there exists any node k, $s < k < t$. By Lemma 3, $RMOST(s) \leq RMOST(k) \leq RMOST(t)$. It follows $RMOST(s) = RMOST(k) = RMOST(t)$. Therefore, vertex k is also a child of vertex j.

Hence, i is the leftmost child of its parent if and only if $RMOST(i-1) < RMOST(i)$. $\square$

Let Q be an array such that $Q(i) = i$ if i is the leftmost child of its parent, otherwise, $Q(i) = 0$. Perform a prefix maxima computation on the array Q and let the resulting array be array $PrefixMaxQ$. Construct an array $Order$ such that $Order(i) = 1 + i - PrefixMaxQ(i)$. It follows $Order(i)$ is the rank of i among its siblings.

Let $CHI(i)$ denote the children of node i in T_{RMOST}. Note that the children of each internal node in T_{RMOST} occupy consecutive ranges. By Lemma 4, $CHI(i)$ can be found in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM. By $RMOST$, CHI and $Order$, we have the tree T_{RMOST} represented by the "parent-of" relation with explicit ordering of children. With the arrays $RMOST$ and $Order$, we can compute $PreOrder(i)$ and $L(i)$ for all i in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM by utilizing the Euler tour technique [1].

Due to the above discussion, we have the following lemma immediately.

Lemma 5 *For an interval graph, its corresponding arrays $RMOST$, $LMOST$, $RSEC$, L and $PreOrder$ can be computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.* $\square$

3 The All-pair Shortest Path Query Problem on Interval Graphs

In this section, we will show how to use data structures described in the previous section to query the length between two given vertices of an interval graph in constant time using a single processor.

Suppose we already construct arrays described in Section 2. Given two interval I_i and I_j, how can we query the length between I_i and I_j ? Without loss of generality, suppose $i < j$. Let $Len(i, j)$ denote the length between I_i and I_j. There are only three cases: 1)$Len(i, j) = 1$, 2)$Len(i, j) = 2$ and 3)$Len(i, j) > 2$. First, we test whether $Len(i, j) = 1$ or not in constant time. Second, if $Len(i, j) \neq 1$, we try to test whether $Len(i, j)$

is equal to 2 or not. By the following lemma, we can decide whether $Len(i,j) = 2$ efficiently.

Lemma 6 *For any two intervals I_i and I_j, $i < j$, $Len(i,j) = 2$ only if $RMOST(i)$ intersects j.*

Proof. Given two intervals, I_i and I_j, $i < j$ and $Len(i,j) = 2$. Note that $b_i < a_j$. Suppose that (I_i, I_s, I_j) is a shortest path between I_i and I_j. That is I_s intersects both I_i and I_j. It follows $a_j < b_s$. Since I_s intersects I_i, by definition of $RMOST(i)$, $b_s \leq b_{RMOST(i)}$. Therefore, $a_j < b_{RMOST(i)}$. It follows that $RMOST(i)$ intersects j. □

Since $RMOST(i)$ is already computed, by the above lemma, we can decide whether $Len(i,j) = 2$ or not in constant time. Now, consider the case that $Len(i,j) > 2$. We have the following lemma.

Lemma 7 *For any two intervals I_i and I_j, $i < j$, if $Len(i,j) > 2$, then $Len(i,j) = Len(RMOST(i), LMOST(j)) + 2$.*

Proof. Given two intervals, I_i and I_j, $i < j$ and $Len(i,j) > 2$. We first prove that $Len(i,j) = Len(RMOST(i), j) + 1$. Let $(I_i, I_k, I_t, \ldots, I_j)$ be one of shortest paths between I_i and I_j. It is easy to see that $Len(I_i, I_t) = 2$. By Lemma 6, we can easily see that $(I_i, I_{RMOST(i)}, I_t, \ldots, I_j)$ is also a shortest path between I_i and I_j. That is $Len(i,j) = Len(RMOST(i), j) + 1$. Similarly, we can prove that $Len(i,j) = Len(i, LMOST(j)) + 1$. Therefore, $Len(i,j) = Len(RMOST(i), LMOST(j)) + 2$. □

An interval is called *proper* if it is not contained by any other interval. It follows for any two proper intervals I_i and I_j, we can have $a_i < a_j$ if and only if $b_i < b_j$. It is easily to prove that for any i, $1 \leq i \leq n$, $I_{LMOST(i)}$ and $I_{RMOST(i)}$ are proper intervals. We have the following lemma.

Lemma 8 *For any interval I_i, $I_{LMOST(i)}$ and $I_{RMOST(i)}$ are proper intervals.* □

Since $I_{LMOST(i)}$ and $I_{RMOST(i)}$ are proper for any i, by Lemma 7, we know that if we can answer the query between two proper intervals then we can answer the query between any two intervals. In the rest of this section, we will show how to answer the query between two proper intervals. The following lemmas and theorem are useful to our query method.

Lemma 9 *For any two intervals I_i and I_j, if $L(i) > L(j)$ and $PreOrder(i) < PreOrder(j)$, then $i < j$.*

Proof. Given two intervals I_i and I_j, suppose $L(i) > L(j)$ and $PreOrder(i) < PreOrder(j)$. Since $L(i) > L(j)$, there exists interval I_t such that $L(t) = L(j)$ and I_t is an ancestor of I_i in T_{RMOST}. Since I_t is an ancestor of I_i in T_{RMOST}, by definition of T_{RMOST}, we have $i < t$ and $PreOrder(t) < PreOrder(i)$. Therefore, $PreOrder(t) < PreOrder(j)$. Since $L(t) = L(j)$ and $PreOrder(t) < PreOrder(j)$, by definition of T_{RMOST}, $t < j$. Therefore, $i < t < j$. □

Lemma 10 *For any two proper intervals I_i and I_j, if $L(i) = L(j)$, then I_i and I_j are adjacent.*

Proof. For two proper intervals I_i and I_j, suppose $L(i) = L(j)$. Without loss of generality, let $i < j$. Let $k = RMOST(i)$. By Lemma 8, I_k is proper. Note that I_k is with a smaller level than I_i. Since I_i and I_j are at the same level on tree T_{RMOST}, I_k is with a smaller level than I_j. By the property of $RMOST$ function, $k > j$. Since I_j is proper, it follows $a_k > a_j$. Since $k = RMOST(i)$, I_k and I_i are adjacent. Besides, I_k and I_i are proper intervals. It follows $b_i > a_k$. Hence $b_i > a_j$. Therefore, I_i and I_j are adjacent. □

Lemma 11 *For any two proper intervals I_i and I_j, if $L(i) > L(j)$ and $PreOrder(i) < PreOrder(j)$, then I_i and I_j are not adjacent.*

Proof. For any two proper intervals I_i and I_j, suppose $L(i) > L(j)$ and $PreOrder(i) < PreOrder(j)$. It follows $i < j$ and $L(RMOST(i)) \geq L(j)$. By definition of T_{RMOST}, $PreOrder(RMOST(i)) < PreOrder(i)$. Therefore, it follows $PreOrder(RMOST(i)) < PreOrder(j)$. Since $L(RMOST(i)) > L(j)$ and $PreOrder(RMOST(i)) < PreOrder(j)$, by Lemma 9, $RMOST(i) < j$. Assume that I_i and I_j are adjacent. We have $RMOST(i) \geq j$ — a contradiction. Hence I_i and I_j are not adjacent. □

Lemma 12 *For any two proper intervals I_i and I_j, if $L(RMOST(i)) = L(j)$ and $PreOrder(i) > PreOrder(j)$, then I_i and I_j are adjacent.*

Proof. For any two proper intervals I_i and I_j, suppose $L(RMOST(i)) = L(j)$ and $PreOrder(i) > PreOrder(j)$. If $RMOST(i) = j$, it is obvious that I_i and I_j are adjacent. Now, consider the case that $RMOST(i) \neq j$. Since $L(RMOST(i)) = L(j)$ and $PreOrder(i) > PreOrder(j)$, by definition of T_{RMOST}, we have $PreOrder(RMOST(i)) > PreOrder(j)$ and then

$RMOST(i) > j$. By Lemma 8, $I_{RMOST(i)}$ is proper. Since $I_{RMOST(i)}$ and I_j are proper, it follows $a_{RMOST(i)} > a_j$. Now, we have $b_i > a_{RMOST(i)} > a_j$. Hence I_i and I_j are adjacent. □

Theorem 13 *For any proper intervals I_i and I_j, $i < j$,*

$$Len(i,j) = \begin{cases} L(i) - L(j) + 1, \\ \quad \textit{if } PreOrder(i) < PreOrder(j), \\ L(i) - L(j), \quad \textit{otherwise.} \end{cases}$$

Proof. Given two proper intervals I_i and I_j, $i < j$. Consider the case that $L(i) = L(j)$. It follows $PreOrder(i) < PreOrder(j)$. By Lemma 10, $Len(i,j) = 1$. Hence $Len(i,j) = L(j) - L(i) + 1$.

Now, consider the case that $L(i) \neq L(j)$. It follows $L(j) < L(i)$. For ease of reference, let $RMOST^k(i)$ denote $RMOST(RMOST^{k-1}(i))$ and $RMOST^1(i) = RMOST(i)$. First, consider the case that $PreOrder(i) < PreOrder(j)$. We claim that the path $P_1 = (i, RMOST(i), \ldots, RMOST^{(L(i)-L(j))}(i), j)$ is a shortest path connecting i and j. Note that $PreOrder(RMOST(i)) < PreOrder(i)$. By the way, we have $PreOrder(i) < PreOrder(j)$. It follows that $PreOrder(RMOST^{(L(i)-L(j)-1)}(i)) < \ldots < PreOrder(RMOST^2(i)) < PreOrder(RMOST^1(i)) < PreOrder(i)$ ¡ $PreOrder(j)$. By Lemma 11, any of intervals, $i, RMOST^k(i)$, for $k = 1, L(i) - L(j) - 1$, is not adjacent to I_j. Since $I_{RMOST^{(L(i)-L(j))}(i)}$ and I_j are at the same level, by Lemma 10, they are adjacent. Hence P_1 is a shortest path connecting intervals I_i and I_j and $Len(i,j) = L(i) - L(j) + 1$.

Second, consider the case that $PreOrder(i) \geq PreOrder(j)$. We claim that the path $P_2 = (i, RMOST(i), \ldots, RMOST^{(L(i)-L(j)-1)}(i), j)$ is a shortest path connecting i and j. Note that any interval I_k where $L(k) < L(j) - 1$ must not be adjacent to I_j. By Lemma 12, $RMOST^{(L(i)-L(j)-1)}(i)$ and j are adjacent. Hence P_2 is a shortest path connecting intervals I_i and I_j and $Len(i,j) = L(i) - L(j)$. □

By Lemma 7 and Theorem 13, we now show our query in the following.

Therefore, we have the following corollary.

Corollary 14 *Given the interval model $F = \{I_1, I_2, \ldots, I_n\}$ of a interval graph G with n vertices such that the intervals in F have been labelled such a way that $b_i < b_j$ if and only if $i < j$, our preprocessing algorithm runs in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM. There are $O(n)$ space data structure computed by the preprocessing algorithm and using those data structure, any shortest length query between two vertices can be answered in $O(1)$ time using one processor.* □

4 The Shortest Path Query Problem on Circular-arc Graphs

In this section, we show how to extend our results on interval graphs to circular-arc graphs. Circular-arc graphs arise in many applications [8]. The definition of circular-arc graphs is the same as that of interval graphs, with the exception that the set of intervals on the real line is replaced by a set of circular-arcs on a unit circle C.

The circular-arc model of a circular-arc graph consists of a set $S = \{I_1, I_2, \ldots, I_n\}$ of n circular-arcs on the unit circle C.

Without loss of generality, we assume that the union of all arcs is equal to C (otherwise, the problem becomes one on interval graphs). Also, we assume that no arc in S covers entire C (otherwise, the shortest length query becomes trivial). We define $I_i = [a_i, b_i]$ is the arc on C from a_i clockwise to b_i. We also assume that the endpoints of the arcs in S are given in the order in which their b_i's points are visited during the clockwise traversal along C by starting at b_1. Without loss of generality, we assume all endpoints are distinct with coordinates of consecutive integer values $1, 2, \ldots, 2n$. Besides, for ease of reference, we assume the coordinate of a_1 is equal to 1. Such labelling can be easily obtained from the sorted array of S in $O(\log n)$ time using $n/\log n$ processors by doing parallel prefix [1].

Given two query arcs I_i and I_j, we assume that I_i does not intersect I_j (otherwise, the problem becomes trivial). Without loss of generality, we assume $b_i < b_j$. The shortest path between I_i and I_j is either the shortest path goes from I_i clockwise to I_j or the shortest path goes from I_i counter clockwise to I_j.

First, we show how to map S into an interval model F such that the shortest path going from I_i clockwise to I_j can be answered easily. This mapping is done as if circle C is open at a_1 and unrolled onto the real line. For every arc $I_k \in S$, if the interior of I_k does not contain a_1, then arc I_k is mapped to the interval $[a_k, b_k]$, which is denoted as J_k. If the interior of I_k contains a_1, I_k is mapped into two intervals: $J'_k = [a_k - 2n, b_k]$ and $J''_k = [a_k, b_k + 2n]$. For example, sup-

Function QUERY(i,j)
if $(i,j) \in E$ then return 1.
if $(RMOST(i), j) \in E$ then return 2.
if $PreOrder(RMOST(i)) < PreOrder(LMOST(j))$ then
return $L(RMOST(i)) - L(LMOST(j)) + 3$
else
return $L(RMOST(i)) - L(LMOST(j)) + 2$
**** $Len(i,j) = Len(RMOST(i), LMOST(j)) + 2$
End of QUERY

Figure 3: The query phase for two vertices on interval graphs.

pose $S = \{[1,5],[4,8],[6,9],[7,11],[10,2],[12,3]\}$. Then, $F = \{[-2,2],[0,3],[1,5],[4,8],[6,9],[7,11],[10,14],[12,15]\}$.

Note that all $J'_k s$ and $J"_k s$ can be found by checking every endpoint in S to see whether it is an endpoint of an arc that contains a_1. This can be done in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.

The following lemma is straightforward and useful at the query phase.

Lemma 15 *For any two non-adjacent arcs I_i and I_j, if $b_i < b_j$, then I_j does not contain a_1.* □

Now, for query arcs I_i and I_j, we can choose corresponding interval on F, such that we can get proper answer from F. Since $b_i < b_j$ and I_i does not intersect I_j, the interior of I_j does not contain a_1. It follows the corresponding interval of I_j on F is J_j. If the interior of I_i does not contain a_1, its corresponding interval on F is also J_i. If the interior of I_i contains a_1, we choose J'_i on F as its corresponding interval. By this way, the shortest path going from I_i clockwise to I_j on S can be answered by querying the shortest path between corresponding intervals on F. In short, we have the following lemma.

Lemma 16 *Given a circular-arc model S and its corresponding interval model F, for any non-adjacent arcs I_i and I_j in S, $b_i < b_j$, the following statements hold.*

1. *If I_i does not contain a_1, then the length of the shortest path going from I_i clockwise to I_j on S is equal to the shortest path length between J_i and J_j on F.*

2. *If I_i contains a_1, then the length of the shortest path going from I_i clockwise to I_j on S is equal to the shortest path length between J'_i and J_j on F.* □

Furthermore, we consider the shortest path going from I_j clockwise to I_i. There are only two possible cases: 1) the interior of I_i contains a_1, 2) the interior of I_i does not contain a_1.

Now, consider the case that the interior of I_i contains a_1. It is not difficult to see that the shortest path from I_j clockwise to I_i on S is equal to the shortest path from J_j to $J^{"}_i$ on F.

We consider the case that the interior of I_i does not contain a_1. We add more intervals into F. For every arc $I_k \in S$, if the interior of I_k does not contain a_1, then the interval $[a_k + 2n, b_k + 2n]$ is added into F. For ease of reference, the corresponding interval is denoted as $\hat{J}_k$. Besides, we denote the resulting interval model F'. It is not difficult to see that the shortest path from I_j clockwise to I_i on S is equal to the shortest path from J_j to $\hat{J}_i$ on F'. In short, we have the following lemma.

Lemma 17 *Given a circular-arc model S and its corresponding interval model F, for any non-adjacent arcs I_i and I_j in S, $b_i < b_j$, the following statements hold.*

1. *If I_i does not contain a_1, then the length of the shortest path going from I_i counter clockwise to I_j on S is equal to the shortest path length between J_j and $\hat{J}_i$ on F'.*

2. *If I_i contains a_1, then the length of the shortest path going from I_i clockwise to I_j on S is equal to the shortest path length between J_j and J'_i on F'.* □

For sake of readability, we discuss how to construct F and F' separately. In fact, since F is a

Algorithm PREPROCESS_circular(S)
For every $I_k \in S$
If the interior of I_k contains a_1
add $J'_k = [a_k - 2n, b_k]$ and $J''_k = [a_k, b_k + 2n]$ into F'
else
add $J_k = [a_k, b_k]$ and $\hat{J}_k = [a_k + 2n, b_k + 2n]$ into F'
end_if
end_for
Relabel the elements in F'.
Build data structure on F' for future query.
End of PREPROCESS_circular

Figure 4: The preprocessing algorithm on circular-arc graphs.

subset of F', we can directly construct F'. Note that we need to relabel elements in F'. Since F' is concatenated by $J'_k s$, $J_k s$, $J''_k s$ and $\hat{J}_k s$, we can relabel them in $O(\log n)$ time using $O(n/\log n)$ EREW processors. We now show our preprocessing algorithm on circular-arc graphs in Figure 4.

Hence, we have shown how to reduce the all-pair shortest path query problem on circular-arc graphs to the same problem on interval graphs. Therefore, we have the following corollary.

Corollary 18 *Given the circular-arc model $S = \{I_1, I_2, \ldots, I_n\}$ of a circular-arc graph G with sorted order, our preprocessing algorithm runs in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM. There are $O(n)$ space data structure computed by the preprocessing algorithm and using those data structure, any shortest length query between two vertices can be answered in $O(1)$ time using single processor.* □

5 Finding All Hinge Vertices on Interval Graphs

In [9], Honma and Masuyama found following lemmas which are very useful for finding all hinge vertices on interval graphs. For any interval I_j in an interval graph, let $D(j)$ denote the set of endpoints between $b_{RSEC(j)}$ and $b_{RMOST(j)}$. That is $D(j) = \{l | b_{RSEC(j)} < l < b_{RMOST(j)}\}$.

Lemma 19 *[9] For any interval I_i in an interval graph, I_i is a hinge vertex if and only if there exist two vertices I_j and I_k with $j < k$, $i = RMOST(j)$ and $a_k \in D(j)$.* □

For example, consider Figure 1 again. $RMOST(3) = 7$,$RSEC(3) = 6$, $b_6 = 14$, $b_7 = 17$. Therefore, $D(3) = \{15, 16\}$. Since $a_9 = 15$, it follows vertex I_7 is the only vertex adjacent both I_3 and I_9 and is a hinge vertex in the interval graph.

Lemma 20 *[9] For any interval I_i and I_j in an interval graph, either $RMOST(i) = RMOST(j)$ or $D(i) \bigcap D(j) = \phi$.* □

By Lemma 3, we have the following lemma.

Lemma 21 *For intervals I_j and I_k in an interval graph, $j < k$, if $RMOST(j) = RMOST(k)$, then $D(k) \subseteq D(j)$.* □

For interval I_i, we test if I_i is the $RMOST$ value of some I_j and there exists a left endpoint in $D(j)$. By Lemma 19, if such I_j exists, I_i is a hinge vertex. Suppose $i = RMOST(j) = RMOST(k)$ and $j < k$. By Lemma 21, if there exists a left endpoint in $D(k)$, there also exists a left endpoint in $D(j)$. Therefore, by Lemma 21, for I_i, among all vertices in $\{j | i = RMOST(j)\}$, we only need to check the smallest one. More precisely, let s be the smallest vertex in the set $\{j | i = RMOST(j)\}$. We only need to check if there is any left endpoint in $D(s)$. If there exists such vertex, vertex i is a hinge vertex. Note that instead of finding all elements in $D(i)$, we can use the starting and ending points to represent set $D(i)$. For ease of reference, let $D(i).l$ and $D(i).r$ denote the starting and ending point respectively. That is, $D(i).l = b_{RSEC(i)} + 1$ and $D(i).r = b_{RMOST(i)} - 1$.

We use an auxiliary array to test if there exists a left endpoint in $D(j)$. Let $LEFT$ be an array such that $LEFT(k) = 1$ if k is a left endpoint, otherwise $LEFT(k) = 0$. By definition, there exists a left endpoint in $D(j)$, if and only if $\sum_{k=D(i).l}^{D(i).r} LEFT(k) > 0$.

Algorithm Hinge_interval(F)
Step 1. Compute array $RMOST$,$RSEC$.
Step 2. For every endpoint j do
if j is a left endpoint then $LEFT(j) := 1$ otherwise $LEFT(j) := 0$
Step 3. For all I_i, $i = 1$ or $RMOST(i) \neq RMOST(i-1)$ do
Step 3.1 $D(i).l := b_{RSEC(i)} + 1$ and $D(i).r := b_{RMOST(i)} - 1$
Step 3.2 Compute $\sum_{k=D(i).l}^{D(i).r} LEFT(k)$,
if its value is not zero, then output I_i as a hinge vertex.
End of Hinge_interval

Figure 5: The algorithm for finding all hinge vertices on interval graphs.

Now, in Figure 5, we show how to compute all hinge vertices on an interval graph step by step.

By Lemma 5, Step 1 can be done in $O(\log n)$ time by using $O(n/\log n)$ processors on EREW PRAM. Obviously, Step 2 can be done with the same processor and time bound.

Now consider Step 3. Note that by Lemma 4, $i > 1$, I_i is the smallest vertex among intervals whose $RMOST$ function value is $RMOST(i)$ if and only if $RMOST(i) \neq RMOST(i-1)$. By Lemma 20 and Lemma 21, all D's visited in this step are mutual disjointed. Therefore, there is no read conflict in Step 3. We can apply parallel prefix sum computation in [1] to perform Step 3.2. For every i in Step 3.2, it takes $O(D(i).r - D(i).l + 1)$ operations. Since all D's visited are mutual disjointed, totally, it needs only $O(n)$ operations.

Therefore, we have the following corollary.

Corollary 22 *Given the interval model F of an interval graph G with sorted order, their all hinge vertices can be found in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.* □

6 Conclusion

In this paper, we propose some optimal parallel algorithms for shortest path related problems on interval and circular-arc graphs. We define some useful data structures on interval graphs. Using our data structure, Honma et al. develop an $O(\log n)$ time optimal parallel algorithm for finding all hinge vertices of a circular-arc graph [10]. These data structures may help to design parallel algorithms for other problems, like center and median problems [3], on interval and circular-arc graphs. The extension to trapezoid graphs [7] is left for future study.

References

[1] S.G. Akl. *Parallel computation: models and methods.* Prentice Hall, Upper Saddle River, New Jersey, 1997.

[2] O. Berkamn and U. Vishkin. Finding level-ancestors in trees. *J. Comput. System Sci.*, 48:214–230, 1994.

[3] S. Bespamyatnikh, B. Bhattacharya, J. Mark Keil, D. Kirkpatrick, and M. Segal. Efficient algorithms for centers and medians in interval and circular-arc graphs. In *Proc. of the 8th Annual European Symposium on Algorithms*, pages 100–111, 2000.

[4] D. Breslauer. The suffix tree of a tree and minimizing sequential transducers. *Theoretical Computer Science*, 191:131–144, 1998.

[5] D.Z. Chen, D. T. Lee, R. Sridhar, and C. N. Sekharan. Solving the all-pair shortest path query problem on interval and circular-arc graphs. *Networks*, pages 249–257, 1998.

[6] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progression. *Proc. of the 19th ACM Symposium on Theory of Computing*, pages 1–6, 1987.

[7] I. Dagan, M.C. Golumbic, and R.Y. Pinter. Trapezoid graphs and their coloring. *Discr. Applied Math.*, 21:35–46, 1988.

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

[9] H. Honma and S. Masuyama. A parallel algorithm for finding all hinge vertices of an interval graph. *IEICE Transactions on Information and Systems*, E84-D(3):419–423, 2001.

[10] H. Honma, S. Masuyama, and F. R. Hsu. An optimal parallel algorithm for finding all hinge vertices of an circular-arc graph. *manuscript*.

[11] Y. L. Wang J. M. Chang, C. C. Hsu and T. Y. Ho. Finding the set of all hinge vertices for strongly chordal graphs in linear time. *Information Sciences*, 99:173–182, 1997.

[12] S. Olariu, J. L. Schwing, and J. Zhang. Optimal parallel algortihms for problems modeled by a family of intervals. *IEEE Trans. Parallel Distrib. Syst.*, (3):364–374, 1992.

[13] R. Seidel. On the all-pair-shortest-path problem. *J. Comput. System Sci.*, 51:400–403, 1995.

[14] R.E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM J. on Computing*, 14(4):862–874, 1985.