Parallel Algorithms for Finding the Center and the median of Interval and Circular-arc Graphs

F. R. Hsu * and M. K. Shan **

Department of Information Technology,
Taichung Healthcare and Management University,
Taiwan, Republic of China

Abstract. The center problem of a graph is motivated by a number of facility location problems. In this paper, we propose parallel algorithms for finding the center of interval graphs and circular-arc graphs. We also consider the median problem of the interval graph. Our algorithms run in $O(\log n)$ time algorithm using $O(n/\log n)$ processors while the intervals and arcs are given in sorted order. Our algorithms are on the EREW PRAM model.

keywords: Parallel Algorithms, EREW PRAM, Center Problem, Interval Graph, Circular-arc Graph.

1 Introduction

A graph $G = (V, E)$ is an *interval graph* if its vertices can be put in a one-to-one corresponded with a set F of intervals on a real line such that two vertices are adjacent in G if and only if their corresponding intervals (circular-arcs) have nonempty intersection. Such a set F is called an *interval* model of the interval graph G. 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. 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 [9].

The center problem of a graph is motivated by a number of facility location problems. For the interval graph, Olariu proposed an $O(|V| + |E|)$ algorithm to compute its 1-center [10]. In [3], Bespamyatnikh et al. proposed an $O(pn)$ time algorithm, where n is the number of vertices, for the p-center problem on a circular-arc graph. Yet, their algorithm does not work while $p = 1$. In this paper, we consider the 1-center problem for both interval and circular-arc graphs. Once the interval and arcs are given in sorted order, our algorithms run in $O(\log n)$ time to find the center of interval and circular-arc graphs using $O(n/\log n)$ EREW PRAM processors.

* Corresponding Author, Department of Information Technology, Taichung Healthcare and Management University, Taiwan, ROC `frhsu@thmu.edu.tw`
** Department of Computer Science, National Chengchi University, Taiwan, R.O.C.

The 1-median problem is to find a vertex such that the sum of the distances to the remaining vertices is minimized. In [3], Bespamyatnikh et al. proposed an $O(n)$ time algorithm for the median problem on interval graphs with sorted intervals. We propose an $O(\log n)$ time algorithm using $O(n/\log n)$ EREW PRAM processors for interval graphs.

The rest of this paper is organized as follows. Section 2 describes basic notations and some interesting properties and data structures on interval graphs. Sections 3 gives algorithms for the center problem on interval and circular-arc graphs respectively. Section 4 gives the algorithm for the median problem on interval graphs. Finally, we conclude our results in Section 5.

2 Preliminaries

In this section, we propose how to compute some useful data structures on interval graphs which will be used in our algorithms. Assume that the interval graph is given by its interval model $F = \{I_1, I_2, \ldots, I_n\}$ with sorted order, where $I_i = [a_i, b_i]$. Let M be the sorted array containing the endpoints of the intervals in F. In case there is no confusion, we also use i to denote interval I_i. 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 $i < j$. By doing parallel prefix computation [1], such labelling can be easily obtained from the sorted array of F in $O(\log n)$ time using $n/\log n$ processors. Since all 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. Our algorithms can be easily 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. In [4], Chao et al. defined similar functions and using these functions to define a successor tree. For each interval I_i, among intervals intersect I_i, consider intervals with rightmost and leftmost endpoints respectively. Formally, let $RMOST(i) = \max\{j | I_j$

contains $b_i\}$ and $LMOST(i) = k$ where a_k is equal to $\min\{a_j | I_j$ contains $a_i\}$. For example, in Figure 1, the array $RMOST[1, \ldots, n]$ is equal to $(4, 4, 6, 8, 9, 9, 9, 11, 11, 11, 11)$. Besides, the array $LMOST[1, \ldots, n]$ is equal to $(1, 1, 1, 1, 4, 3, 4, 4, 6, 8, 8)$.

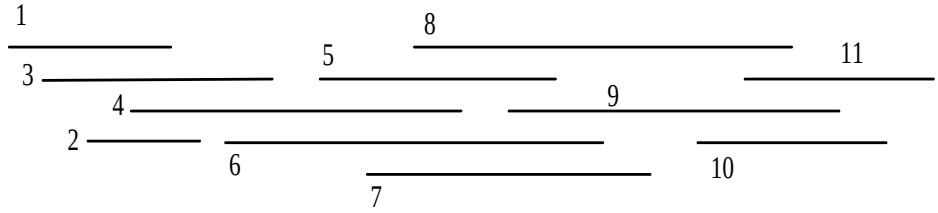

Fig. 1. A set of intervals.

According to the $RMOST$ array, the successor tree T_{RMOST} is defined 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 $i < j$. Consider Figure 1 again. Its corresponding T_{RMOST} is shown in Figure 2. Let $PreOrder(i)$ and $LEV(i)$ denote the pre-order number and the level of interval i in tree T_{RMOST}

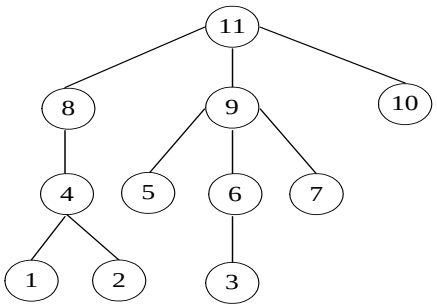

Fig. 2. The tree T_{RMOST}.

respectively. In this example, the pre-order traversal of T_{RMOST} would be $(11, 8, 4, 1, 2, 9, 5, 6, 3, 7, 10)$ and $PreOrder = (4, 5, 9, 3, 7, 8, 10, 2, 6, 11, 1)$ and $LEV = (3, 3, 3, 2, 2, 2, 2, 1, 1, 1, 0)$. Chao et al. showed that these data structure can be found efficiently.

Lemma 1. *[4] For an interval graph, its corresponding arrays RMOST, LMOST, LEV and PreOrder can be computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.* □

An interval is called *proper* if it is not contained by any other interval. Let $LEN_F(i,j)$ denote the shortest path length between I_i and I_j on F. The following lemmas show how to query the length between I_i and I_j.

Lemma 2. *[4] For any two intervals I_i and I_j, $i < j$, if $LEN_F(i,j) > 2$, then $LEN_F(i,j) = LEN_F(RMOST(i), LMOST(j)) + 2$.* □

Lemma 3. *[4]For any proper intervals I_i and I_j, $i < j$,*

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

For each interval I_i, consider intervals on its right side. Let $RminB(i)$ denote the interval with the minimum right endpoint. If no such interval exists, let $RminB(i) = n + 1$. Formally, $RminB(i) = \min(\{j | a_j > b_i\} \bigcup \{n+1\})$. For example, consider Figure 1. The array $RminB[1, \ldots, n]$ is equal to $(5, 5, 5, 9, 10, 10, 10, 12, 12, 12, 12)$. Using the list of all intervals in F sorted by the $a_i's$, we can apply the parallel prefix computations [1] to compute the array $RminB$ in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model. Therefore, we have the following lemma

Lemma 4. *For an interval graph, its corresponding array RminB can be computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.* □

3 Finding the Center

The p-center problem is to locate p facilities on a graph so as to minimize the largest distance between the other vertices and the p facilities.

For the interval graph, Olariu proposed an algorithm linear to numbers of vertices and edges to compute its 1-center [10]. In [3], Bespamyatnikh et al. proposed an $O(pn)$ time algorithm for the p-center problem on a circular-arc graph. Yet, their algorithm does not work while $p = 1$. In this paper, we consider the 1-center problem for both interval and circular-arc graphs. We propose $O(\log n)$ time algorithms using $O(n/\log n)$ EREW PRAM processors for interval and circular-arc graphs.

3.1 The Center Problem on interval graphs

Now consider the 1-center problem for interval graphs. Given the interval model F, with sorted endpoints, let $r(F)$ denote the *radius* of the interval graph. That is

$$r(F) = \min_{i \in F} \max_{j \in F} Len(i, j).$$

Therefore, k is the center of F, if and only if

$$\max_{j \in F} Len(k, j) = r(F).$$

Note that the distance between the interval with the leftmost right endpoint and the interval with the rightmost left endpoint is the maximum distance between any two intervals. Therefore we have the following lemma.

Lemma 5. *Given an interval model F, suppose I_k is the interval with largest left endpoint and $d = Len(1, k)$. Then $r(F) = \lceil d/2 \rceil$ and $RMOST^{\lceil d/2 \rceil}(1)$ is the center of the interval graph.*

Proof. Suppose I_k is the interval with largest left endpoint and $d = Len(1, k)$. Let $t = \lceil d/2 \rceil$. Note that $Len(1, RMOST^t(1)) = t$ and $Len(RMOST^t(1), k) = d - t = \lfloor d/2 \rfloor$. It is easy to see that $\max_{j \in F} Len(RMOST^t(1), j) = t$. Suppose that I_q is the center. It follows that $Len(1, q) \leq r(F)$ and $Len(q, k) \leq r(F)$. Therefore, $Len(1, k) \leq Len(1, q) + Len(q, k) \leq 2 * r(F)$. Hence $Len(1, k) \leq 2 * r(F)$. Assume that $r(F) < t$. That is $r(F) \leq t - 1$. Hence $d = Len(1, k) \leq 2(t - 1)$ — a contradiction. Therefore $r(F) \geq t$. Since $\max_{j \in F} Len(RMOST^t(1), j) = t$, we have $r(F) = t$ and $RMOST^t(1)$ is the center of the interval graph. □

The following lemma shows how to find $RMOST^t(i)$ for any interval I_i efficiently.

Lemma 6. *For any non-root interval I_i, $RMOST^t(i) = \max\{j | PreOrder(j) < PreOrder(i)$ and $L(j) = L(i) - t\}$.*

Proof. For any non-root interval I_i, by definition, $RMOST^t(i)$ is at level $L(i) - t$ in T_{RMOST}. Since $RMOST^t(i)$ is an ancestor of i in T_{RMOST}, $PreOrder(RMOST^t(i)) < PreOrder(i)$. By definition,

the pre-order numbers of $RMOST^t(i)$'s siblings on its left side are less than $PreOrder(RMOST^t(i))$. Besides, the pre-order numbers of $RMOST^t(i)$'s siblings on its right side are greater than $PreOrder(i)$. Therefore, $RMOST^t(i) = \max\{j | PreOrder(j) < PreOrder(i) \text{ and } L(j) = L(i) - t\}$. □

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

Lemma 7. *For any two intervals I_i and I_j, if $i < j$, then $RMOST(i) \leq RMOST(j)$.*

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, $RMOST(i) \leq RMOST(j)$. □

By Lemma 7, we observe that the children of each internal node in T_{RMOST} occupy consecutive ranges. For example, in Figure 2, the children of node 9 are from node 5 to node 7. 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 8. *In T_{RMOST}, the children of each internal node in T_{RMOST} occupy consecutive ranges. Besides, for any non-root vertex i,$i > 1$, 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 7, $RMOST(s) \leq RMOST(k) \leq RMOST(t)$. 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 $RMOST(i-1) < RMOST(i)$. □

Since the pre-order number of nodes in the same level are increasing from left to right and by Lemma 5, 6 and 8, we can perform binary search to locate $RMOST^t(i)$ for any interval I_i. Therefore, we have the following corollary.

Corollary 1. *Given the interval model F of an interval graph G with sorted order, the center can be found in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.* □

3.2 The Center Problem on Circular-arc Graphs

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. For example, see Figure 3.

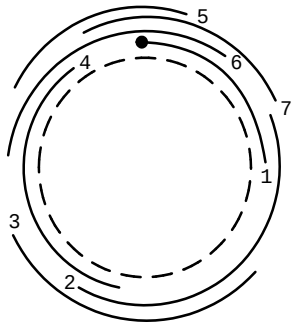

Fig. 3. A circular-arc model on circle C.

Now consider the center problem for circular-arc graphs. Without loss of generality, we assume that the union of all arcs is equal to C (otherwise, the problem becomes one on interval graphs). Besides, we assume that there is no arc equal to C (otherwise, the problem becomes trivial). Given the circular-arc model S, let $r(S)$ denote the *radius* of the circular-arc graph. That is $r(S) = \min_{i \in S} \max_{j \in S} Len(i, j)$. Therefore, k is the center of S, if and only if $\max_{j \in S} Len(k, j) = r(S)$.

Similar to the $RMOST$ function on an interval graph, for an arc I_i on a circular-arc graph, we define $CMOST(i)$ as follows. Let $N(i)$ denote the set $\{I_j | b_i \text{ in } I_j\}$. Starting from b_i, we visit right endpoints of arcs in $N(i)$ clockwise one by one. Let $CMOST(i)$ denote the last arc visited. For example, consider Figure 3. The arrays $CMOST[1, \ldots, n]$ is equal to $(2, 4, 4, 6, 1, 1, 1)$. Similar to the $LMOST$ function on an interval graph, for an arc I_i on a circular-arc graph, we define $DMOST(i)$ as follows. Let $N'(i)$ denote

the set $\{I_j | a_i \text{ in } I_j\}$. Starting from a_i, we visit left endpoints of arcs in $N'(i)$ counter clockwise one by one. Let $DMOST(i)$ denote the last arc visited. Consider Figure 3 again. The arrays $DMOST[1,\ldots,n]$ is equal to $(6,1,2,2,4,4,6)$. For ease of reference, let $CMOST^k(i)$ denote $CMOST(CMOST^{k-1}(i))$ and $CMOST^1(i) = CMOST(i)$. Besides, $CMOST^0(i) = i$. $DMOST^k(i)$ is defined similarly.

In the following, we will show many problems on circular-arc graphs can be transformed into problems on interval graphs. We describe how to map S into an interval model F'. This mapping is done as if circle C is open at a_1 and unrolled onto the real line twice. Any arc I_k is mapped into two intervals J_k^1 and J_k^2 as follows. If the interior of I_k does not contain a_1, then $J_k^1 = [a_k, b_k]$ and $J_k^2 = [a_k + 2n, b_k + 2n]$. If the interior of I_k contains a_1, then $J_k^1 = [a_k - 2n, b_k]$ and $J_k^2 = [a_k, b_k + 2n]$. For example, consider Figure 3. Its corresponding interval model is shown in Figure 4. Note that the mapping can be found

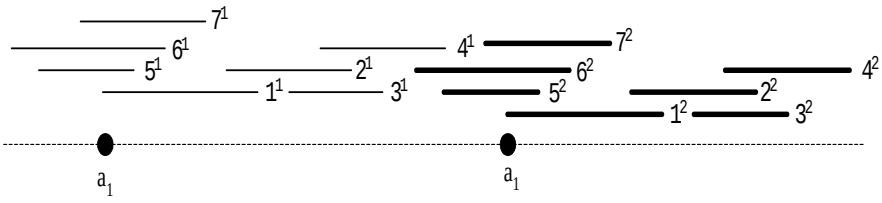

Fig. 4. The corresponding interval model of the circular-arc model in Figure 3.

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 shows how to compute array $CMOST$ of S through the help of F'. Here, for an arc I_i on S, we choose J_i^1 as its corresponding interval on F'. For an interval on F', its corresponding arc on S is the arc which mapped into the interval.

Lemma 9. *Given a circular-arc model S, I_i is an arc on it. Suppose the corresponding interval of I_i on F' is t. Then, $CMOST(i)$ is equal to the corresponding arc of $RMOST(t)$ on F'.* □

Similar to the $RminB$ function on an interval graph, for an arc I_i on a circular-arc graph, we define $CminB(i)$ as follows. Starting from b_i, we visit right endpoints of arcs which do not intersect I_i clockwise. Let $CminB(i)$ denote the first arc visited. In Figure 3, array $CminB = (3,5,5,7,2,2,2)$.

Similar to Lemma 9, for the $CminB$ function, we have the following lemma.

Lemma 10. *Given a circular-arc model S, I_i is an arc on it. Suppose the corresponding interval of I_i on F' is t. Then, $CminB(i)$ is equal to the corresponding arc of $RminB(t)$ on F'.* □

By Lemma 9 and 10, we can compute arrays $CMOST$ and $CminB$ on a circular-arc graph by computing its corresponding arrays $RMOST$ and $RminB$ on its corresponding interval graph. Regarding to $DMOST$, we can apply the techniques on $CMOST$ to find them. By Lemma 1 and 4, we have the following lemma.

Lemma 11. *For a circular-arc graph, its corresponding array $CMOST$, $CminB$ and $DMOST$ can be computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.* □

For arc I_i in S, let $R(i)$ denote the shortest path length walking from I_i clockwise and visiting I_i again. For example, in Figure 3, $(I_1, I_2, I_4, I_6, I_1)$ is a path and $R(1) = 4$. For ease of reference, let $r_i(S)$ denote the largest distance between arc I_i and other arcs. That is $r_i(S) = \max_{j \in S} Len(i, j)$. Suppose $\lfloor R(i)/2 \rfloor = k$. We define the *detecting area* of I_i as $Da(i) = [b_{CMOST^{k-1}(i)}, a_{DMOST^{k-1}(i)}]$. We have the following lemma.

Lemma 12. *For arc I_i, suppose $\lfloor R(i)/2 \rfloor = k$. If there is no arc contained in $Da(i)$, then $r_i(S) = k$. Otherwise $r_i(S) = k + 1$.*

Proof. For arc I_i, suppose $\lfloor R(i)/2 \rfloor = k$. Consider $CMOST^k(i)$. The shortest path length from I_i to $CMOST^k(i)$ clockwise is k. Since the shortest path length walking from I_i clockwise and visiting I_i again is $R(i)$, the shortest path length from I_i to $CMOST^k(i)$ counter-clockwise is at least k. Therefore, $Len(i, CMOST^k(i)) = k$. It follows that $r_i(S) \geq k$. Consider the arcs which are not contained in $Da(i)$. Their distances to I_i are less than or equal to k. If there is no arc contained in $Da(i)$, $CMOST^k(i)$ is the arc with the largest distance between arc I_i and other arcs and $r_i(S) = k$.

Suppose there exits some arcs contained in $Da(i)$. See Figure 5. These arcs intersect $CMOST^k(i)$ or $DMOST^k(i)$. Therefore, their distance to I_i is $k + 1$. Hence, $r_i(S) = k + 1$. □

Note that $r(S) = \min_{i \in S} r_i(S)$. By Lemma 12, we have the following lemma.

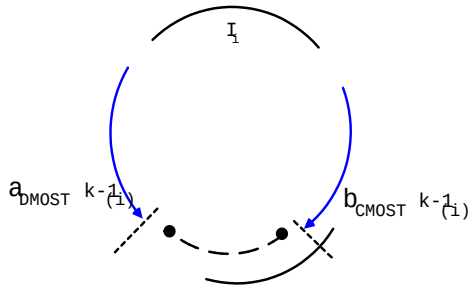

Fig. 5. The illustration of Lemma 12.

Lemma 13. *Given a circular-arc model S, there exists a center I_j such that $R(j) = \min_{i \in S} R(i)$ or $\min_{i \in S} R(i) + 1$.* □

By the above lemma, the center can be found by computing $\min\{r_j(S) | R(j) = \min_{i \in S} R(i)$ or $\min_{i \in S} R(i) + 1, j \in S\}$. Now, we list steps for computing the center of a circular-arc graph.

Step 1. For every arc I_i in S, compute $R(i)$.

Step 2. Find $\min_{i \in S} R(i)$. Let $t = \min_{i \in S} R(i)$. Let the set of arcs $CA = \{I_i | R(i) = t$ or $R(i) = t + 1\}$.

Step 3. For every arc I_j in CA, compute $Da(j)$.

Step 4. For every arc I_j in CA, if there exists no arc in $Da(j)$, then let $r_j(S) = \lfloor R(j)/2 \rfloor$, otherwise let $r_j(S) = \lfloor R(j)/2 \rfloor + 1$.

Step 5. Find the center I_c where $r_c(S) = \min\{r_j(S) | I_j \in CA\}$.

Now, we consider Step 1. Recall that when we map the circular-arc model S into its corresponding interval model F', we map each arc I_i into two intervals, say J_i^1 and J_i^2. It is not difficult to see that $R(i)$ is equal to $Len(J_i^1, J_i^2)$ in F'. We have the following lemma.

Lemma 14. *Given a circular-arc model S and its corresponding interval model F', $R(i)$ is equal to $Len(J_i^1, J_i^2)$ in F'.* □

As described in Section 2, we can use one processor to query $Len(J_i^1, J_i^2)$ in constant time. Note that in order to avoid read conflict, for every interval I_i in F', we need to store $PreOrder(RMOST(i))$, $PreOrder(LMOST(i))$, $L(RMOST(i))$ and $L(LMOST(i))$ for future query during the preprocessing

phase. Therefore, Step 1. can be performed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors. Obviously, Step 2. can be done in the same time and processor complexity.

Regarding Step 3, we need to find $Da(j)$. That is we need to query $CMOST^k(j)$ and $DMOST^k(j)$ where $k = \lfloor R(j)/2 \rfloor - 1$. Note that $CMOST^k(j)$ is equal to $RMOST^k(j)$ in its corresponding T_{RMOST} tree. We can use the technique of the level-ancestor query in trees introduced by Berkman and Vishkin [2] to solve these queries. However, it is a fairly hard implemented algorithm and run on the CREW PRAM.

In stead of answering these queries individually, we perform these queries in batch. With the help of T_{RMOST}, the following lemma shows how to find $CMOST^k(j)$ for all I_j in S for some fixed k.

Lemma 15. *Given a circular-arc model S and a positive integer k, $CMOST^k(i)$ and $DMOST^k(i)$ for all I_i in S can be found in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

Proof. First, we map the circular-arc model S into corresponding interval model F'. Given arc I_i on S, suppose the corresponding interval of I_i on F' is t. By Lemma 9, it follows $CMOST^k(i)$ is equal to the corresponding arc of $RMOST^k(t)$ on F'.

Now, consider how to compute $RMOST^k(t)$ on F' for all t. Note that $RMOST^k(t)$ is the ancestor of t on level $L(t) - k$ in T_{RMOST}. By Lemma 6, we can compute $RMOST^k(t)$ on F' for all node t at level j as follows. We merge the nodes on level $j - k$ and j according to their pre-order number in T_{RMOST}. For nodes on level $j - k$ and on level j, define $JK(t)$ such that $JK(t) = t$ if node t is on level j, otherwise $JK(t) = 0$. Then, the prefix maximum of JK on the merged list is equal to $RMOST^k(t)$ for node t on level j. The merging process for two sorted lists and prefix computation can be performed in $O(\log h)$ time using $O(h/\log h)$ EREW PRAM processors [1, 6], where h is the size of lists. The total size for all levels is at most $2n$. Then, $RMOST^k(t)$ on F' for all t can be computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.

It follows $CMOST^k(i)$ for all arc I_i on S can be found in the same time and processor complexity. The proof of $DMOST^k(i)$ is similar and omitted. □

Suppose $t = \min_{i \in S} R(i)$ and $CA = \{I_i | R(i) = t \text{ or } R(i) = t+1\}$. We can find $Da(j)$ for every arc I_j in CA, by performing batch query (as described in Lemma 15) *twice*, where $k = \lfloor t/2 \rfloor - 1$ and $k = \lfloor (t+1)/2 \rfloor - 1$. Therefore, Step 3 can be done in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.

Now, consider Step 4. We need to test if there exists any arc in $Da(j)$ for arc I_j. The following lemma shows how to perform this test efficiently.

Lemma 16. *For arc I_j in S, there exist any arc contained in $Da(j)$ if and only if $CminB(CMOST^{\lfloor R(j)/2 \rfloor - 1}(j))$ is contained in $Da(j)$.* □

Recall that when we compute $Da(j)$, we store $CMOST^{\lfloor R(j)/2 \rfloor - 1}(j)$ for node j. To avoid read conflict, in Step 3, when we compute $Da(j)$, we can also store $CminB(CMOST^{\lfloor R(j)/2 \rfloor - 1}(j))$ for node j. Therefore, Step 4 be done in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors. Obviously, Step 5 can be performed in the same time and processor complexity. Therefore, we have the following corollary.

Corollary 2. *Given the circular-arc model S of an interval graph G with sorted order, the center can be found in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.* □

4 Finding the Median of an Interval Graph

4.1 Definition

In order to compute some of the structure of F, we define the following notations. For any endpoint q, let $LaNo(q)$ and $LbNo(q)$ denote the number of $a_i's$ and $b_i's$ to the left of q respectively. Similarly, let $RaNo(q)$ and $RbNo(q)$ denote the number of $a_i's$ and $b_i's$ to the right of q respectively. For example, in Figure 1, $a_6 = 7$,$LaNo(7) = 4$. Note that all these arrays can be computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors by parallel prefix computation [1].

The 1-median problem is to find a vertex such that the sum of the distances to the remaining vertices is minimized. For an interval I_i, let $Cost(I_i) = \sum_{I_j \in F} Len(i,j)$ denote the cost of I_i . Since the cost of

a non-proper interval is not smaller than the cost of the proper interval containing it, we can consider only proper intervals as candidates for the median.

For a candidate interval I_i, $Cost(I_i)$ can be divide into two parts. For an endpoint q, which is an endpoint of I_i, we define $LSUM(q) = \sum_{b_j<q} Len(i,j)$ and $RSUM(q) = \sum_{b_j>q} Len(i,j)$. It follows $Cost(I_i) = LSUM(b_i) + RSUM(b_i)$. Therefore, if we can compute $LSUM(q)$ and $RSUM(q)$ for each endpoint of a proper interval, the median can be found efficiently. The following lemmas show key idea about how to compute $LSUM$ and $RSUM$ efficiently.

Lemma 17. *[3] For any proper interval I_i, the following equations hold.*

1. $LSUM(a_i) = LSUM(a_{LMOST(i)}) + 2 * LbNo(a_i) - LbNo(a_{LMOST(i)})$
2. $LSUM(b_i) = LSUM(a_i) + LbNo(b_i) - LbNo(a_i)$
3. $RSUM(a_i) = RSUM(b_i) + LbNo(b_i) - LbNo(a_i)$
4. $RSUM(b_i) = RSUM(b_{RMOST(i)}) + RbNo(b_i) + RaNo(b_i) - RbNo(b_{RMOST(i)})$ □

For any proper interval I_i, let's discuss how to compute its corresponding $RSUM(a_i)$ and $RSUM(b_i)$. The computation for $LSUM(a_i)$ and $LSUM(b_i)$ are similar and omitted. For every interval I_i, let $wa_i = LbNo(b_i) - LbNo(a_i)$ and $wb_i = RbNo(b_i) + RaNo(b_i) - RbNo(b_{RMOST(i)})$.

By Lemma 17, $RSUM(a_i) = RSUM(b_i) + wa_i$. Therefore, we can compute $RSUM(a_i)$ after finding $RSUM(b_i)$ and wa_i. Now, consider the computation method of $RSUM(b_i)$.

By Lemma 17, $RSUM(b_i) = RSUM(b_{RMOST(i)}) + wb_i$. Consider the tree T_{RMOST} again. For every node i in T_{RMOST}, we associate weight wb_i with it. By this way, we get a weighted T_{RMOST}. Note that for any proper interval I_i, $RSUM(b_i)$ is equal to the summation of all weights on the path from node i to the root in T_{RMOST}. Therefor, we can reduce the problem of finding $RSUM(b_i)$ for proper intervals into the following problem.

Prefix sum on weighted T_{RMOST}: For every node I_i in T_{RMOST}, compute the summation of weights in $\{wb_j | j$ is on the path from node i to the root in $T_{RMOST}\}$.

We can say that prefix sum on weighted T_{RMOST} is a generalized prefix sum computation.

Recall that $LbNo$ and $RbNo$ value of all endpoints can be pre-computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors. Therefore, all $wa_i's$ and $wb_i's$ can be computed in the same processors and time bound. Note that when we compute $wb_i's$, we read $RbNo(b_{RMOST(i)})'s$. It may leads to read conflict. In order to avoid read conflict, we construct array BR such that $BR(i)$ stores $RbNo(b_{RMOST(i)})$ as follows. As shown in Lemma 8, the children of each internal node in T_{RMOST} occupy consecutive ranges. Therefore, we can apply prefix computation technique [1] to propagate $RbNo(b_j)$ to j's children in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.

4.2 The Transformation of a General Tree to a Binary Tree

Instead of computing prefix sum on weighted T_{RMOST} directly, we first transform the weighted T_{RMOST} into a weighted binary tree.

Given a sub-tree T, rooted at u and with children $v_1, v_2, \cdots, v_k$, its corresponding transformed binary sub-tree BT is defined as follows and is shown in Fig. 6:

1. For $u, v_1, v_2, \cdots, v_k$ in T, their corresponding nodes in BT are $u, v_1, v_2, \cdots, v_k$, respectively. Besides, their weights are the same as in T.
2. If k=1, there is a dummy node u'_1 in BT. If $k > 2$, there are dummy nodes $u'_1, u'_2, \cdots, u'_{k-2}$ in BT. The weight of any dummy node is zero.
3. The root of BT is u.
4. The parent of v_1 and u'_1 is u.
5. The parent of v_i and u'_i is u'_{i-1} (for i=2 to k-2).
6. The parent of v_{k-1} and v_k is u'_{k-2}.
7. If there is any sub-tree with v_i as its root, apply the above rules recursively to transform it into a binary tree.

The transformation of a tree T to a binary tree BT is now illustrated in Fig. 6. Fig. 6(a) is a general tree. Its corresponding binary tree is in Fig. 6(b). All of the newly added nodes in this binary tree are

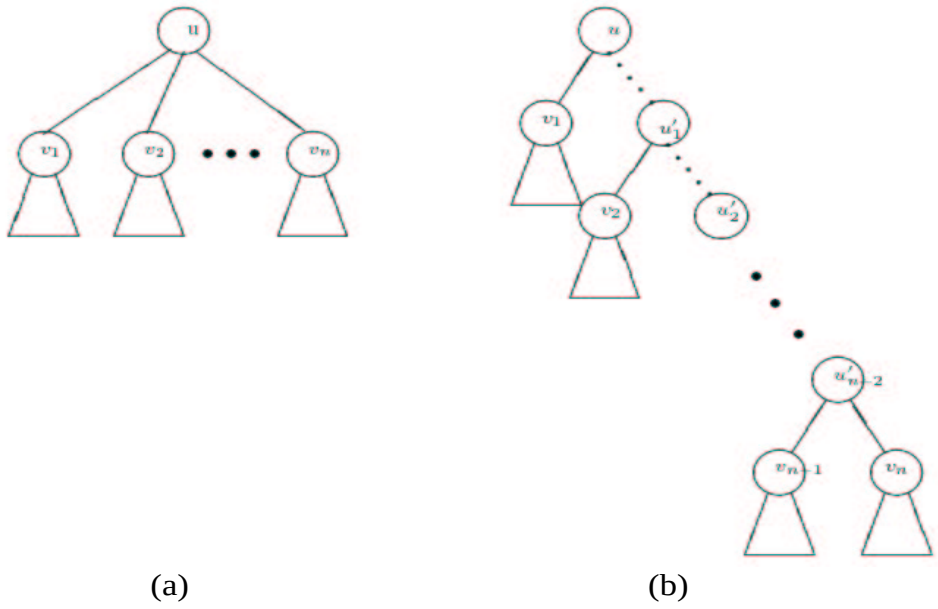

Fig. 6. The General Tree and the Corresponding Binary Tree.

dummy nodes. Throughout this paper, the original general tree is denoted T and its transformed binary tree as BT.

We apply the above transformation process on T_{RMOST}. Let BT_{RMOST} denote the weighted binary tree. With the arrays computed in Section 2, we can perform these transformation in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors. It is not difficult to see that for any non-dummy node in BT_{RMOST}, the summation of weights on the path from it to the root in BT_{RMOST} is equal to the summation of weights on the path from its corresponding vertex to the root in T_{RMOST}. Therefore, we can reduce the problem about the prefix sum on T_{RMOST} into the prefix sum on BT_{RMOST} in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.

The computation of prefix sum on BT_{RMOST} can be considered as computing *downward accumulation* on a binary tree. Downward accumulation passes information downwards, from the root towards the leaves. In [8], Gibbons et al. proposed an $O(\log n)$ time algorithm using $O(n/\log n)$ EREW PRAM processors to solve downward accumulation on a binary tree. Therefore, we have the following corollary.

Corollary 3. *Given the interval model F of an interval graph G with sorted order, the median can be found in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.* □

5 Conclusion

In this paper, we propose parallel algorithms for center and median problems on interval and circular-arc graphs. We define some useful data structures on interval graphs. The median problem on circular-arc graphs is left for future study. The extension to trapezoid graphs [7] is also 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. *Networks*, 39(3):144–152, 2002.
4. H. S. Chao, F. R. Hsu, and R. C. T. Lee. On the shortest length queries for interval and circular-arc graphs. *Proc. of the joint meeting of the Fifth World Multi-conference on Systemics, Cybernetics and Informatics (SCI 2001) and the 7th International Conference on Information Systems Analysis and Synthesis (ISAS 2001), Orlando, USA*, VII:331–336, 2001.
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. R. Cole. Parallel merge sort. *SIAM J. on Computing*, 17:770–785, 1988.
7. I. Dagan, M.C. Golumbic, and R.Y. Pinter. Trapezoid graphs and their coloring. *Discr. Applied Math.*, 21:35–46, 1988.
8. J. Gibbons, W. Cai, and D. Skillcorn. Efficient parallel algorithms for tree accumulations. *Science Computer Programming*, 23:1–18, 1994.
9. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.
10. S. Olariu. A simple linear time algorithm for computing the center of an interval graph. *International Journal of Computer Mathematics*, 34:121–128, 1990.