Efficient algorithms for centers and medians in interval and circular-arc graphs

S. Bespamyatnikh[1], B. Bhattacharya[2], J. Mark Keil[3], D. Kirkpatrick[1], and M. Segal[1]

[1] Department of Computer Science, University of British Columbia, Vancouver, B.C. Canada V6T 1Z4
{besp, kirk, segal}@cs.ubc.ca
[2] School of Computing Science, Simon Fraser University, Burnaby, B.C. Canada, V5A 1S6
binay@cs.sfu.ca
[3] Dept. of Computer Science, University of Saskatchewan, 57 Campus Dr., Saskatoon, Sask., Canada, S7N 5A9
keil@cs.usask.ca

Abstract. The p-center problem is to locate p facilities on a network so as to minimize the largest distance between the n demand points and p facilities. The p-median problem is to locate p facilities on a network so as to minimize the average distance from one of the n demand points to one of the p facilities. We consider the p-center and p-median problems when the network can be viewed as an interval or circular-arc graph. We provide, given the interval model of an n vertex interval graph, an $O(n)$ time algorithm for the 1-median problem on the interval graph. We also show how to solve the p-median problem, for arbitrary p, on an interval graph in $O(pn \log n)$ time and on an circular-arc graph in $O(pn^2 \log n)$ time. Other than for trees, no polynomial time algorithm for p-median problem has been reported for any large class of graphs. We also show how to solve the p-center problem on an circular-arc graph in $O(np)$ time, assuming that the endpoint of the given arcs are sorted. The algorithm is based on the *spring model* of computation.

1 Introduction

The p-center problem is to locate p facilities on a network so as to minimize the largest distance between the n demand points and p facilities. In the p-median problem we endeavour to locate p facilities on a network so as to minimize the average distance from one of n demand points to one of the p facilities. These problems are central to the field of location theory and has been researched extensively [3, 7, 10, 14, 15, 17, 20, 21]. Applications include the location of industrial plants, warehouses, and public service facilities on transportation networks as well as the location of various service facilities in telecommunication networks [3, 10, 15, 17, 21].

We model the network with a graph $G = (V, E)$ on n vertices and assume that the demand points coincide with the vertices. We also restrict the facilities to vertices as Hakimi [10] has shown that for the p-median problem the possible sites for the facilities can always be restricted to the set of vertices without increasing the cost. The p-median problem then becomes that of finding a set $X \subset V$ such that $|X| = p$ and for which $\sum_{i=1}^{n} d(v_i, X)$ is minimum.

If $p = 1$ the problem is known as the 1-median problem and Hakimi [10] optimally solved it in a general network in $O(n^3)$ time. In a tree network Goldman [8] and Kariv and Hakimi [14] derive $O(n)$ time algorithms for the 1-median problem. One can also find the 2-median of a general network in $O(n^3)$ time by considering all possible pairs of vertices as medians. For tree networks Gavish and Sridhar present an $O(n \log n)$ time algorithm [7].

For general p, Kariv and Hakimi [14] showed that the p-median problem is NP-complete. They were, however, able to produce an $O(p^2 n^2)$ time algorithm for the case of tree networks [14]. The tree network algorithm was recently improved to $O(pn^2)$ by Tamir [20]. Other than for trees, no polynomial time algorithm for the p-median has been reported for any large class of graphs. In this paper we provide algorithms for the p-median problem on interval graphs.

The p-center problem for a given graph $G = (V, E)$ (restricting the facilities to be a subset of V) is to find a set $C \subset V$ such that $|C| = p$ and $\max_{i=1}^{n} d(v_i, C)$ is minimized. Regarding this problem Olariu provides a linear time algorithm for locating a single central facility that minimizes the maximum distance to a demand point [19]. Frederickson [5] showed how to solve this problem for trees in optimal linear time (not necessary restricting the facilities to be the vertices of the tree) using parametric search. The work of Kariv and Hakimi [13] presents several results for general graphs; however since the problem is known to be NP-complete they were able to give only $O(n^{2p+1} \log n/(p-1)!)$ runtime algorithm. Some work also has been done for approximating the p-center solution, see e.g, [1].

A graph $G(\mathcal{S}) = (V, E)$ ($G(\mathcal{A}) = (V, E)$) is an interval (circular-arc) graph if there exists a set $\mathcal{S}$ ($\mathcal{A}$) of intervals (arcs) on the real line (unit circle) such that there is a one-to-one correspondence between vertices $v_i \in V$ and intervals (arcs) $I_i \in \mathcal{S}$ such that an edge $(v_i, v_j) \in E$ if and only if $I_i \cap I_j \neq \emptyset$. The set $\mathcal{S}$ ($\mathcal{A}$) is called the interval (circular-arc) model for G. Interval and circular-arc graphs are important tools in many application areas, including scheduling and VLSI layout [9, 18].

The problem of recognizing interval and circular-arc graphs is known to be solved in $O(|V|+|E|)$ time (see e.g, [2], and we assume an interval (circular-arc) model $\mathcal{S}$ ($\mathcal{A}$) for G is available.

In the next section we review some relevant results on interval and circular-arc graphs. Section 3 provides an $O(n)$ time algorithm for the 1-median problem in interval graph. In Section 4 we generalize the result for arbitrary p and give an $O(pn \log n)$ time algorithm. We show how to apply this result in order to solve the p-median problem in circular-arc graphs in Section 5. Section 6 presents an $O(np)$ runtime solution for the p-center problem.

2 Preliminaries

Let $\mathcal{S}$ ($\mathcal{A}$) be the set of n intervals (arcs) in the interval (circular-arc) model for G. Without loss of generality, we assume that all the interval (arc) endpoints are distinct. We define each interval (arc) $I_i \in \mathcal{S}$ ($I_i \in \mathcal{A}$) by its left endpoint a_i and its right endpoint b_i. Once the endpoints are sorted we can replace the real value of an endpoint by its rank in the sorted order. Thus we use the integers from 1 to $2n$ as coordinates for the endpoints of the intervals (arcs) in $\mathcal{S}$ ($\mathcal{A}$). That is, each integer $j \in [1 \ldots 2n]$ is a_i (or b_i) for some interval (arc) $I_i \in S$ ($I_i \in \mathcal{A}$).

From the sorted list of interval endpoints, in $O(n)$ time, we compute the numbers of a's and b's left or right of every point q. In particular, let $\#aL(q)$ (likewise $\#bL(q)$) be the number of left (likewise right) endpoints of intervals in $\mathcal{S}$ that lie to the left of integer q, for $q \in [1 \ldots 2n]$. Similarly define $\#aR(q)$ (and $\#bR(q)$) to be the number of left (or right) endpoints of intervals in $\mathcal{S}$ that lie to the right of q. We use these quantities to quickly compute some of the structure of $\mathcal{S}$. For example, the number of intervals left of interval I_i is $\#bL(a_i)$.

Chen et al [4] define a successor function on intervals (arcs) in their paper on solving the all pairs shortest path problem on interval and circular-arc graphs. We use their idea to define a right successor and a left successor of an integer q. We say $RSUC(q) = I_i \in S$ if and only if $b_i = max\{b_j | I_j \text{ contains } q\}$ and $LSUC(q) = I_i \in S$ if and only if $a_i = min\{a_j | I_j \text{ contains } q\}$. For an interval I_i, $RSUC(I_i) = RSUC(b_i)$, and $LSUC(I_i) = LSUC(a_i)$.

We also define the ith iterated right successor of an integer q $RSUC(q, i)$ to be $RSUC(RSUC($ $\ldots RSUC(q, i)))$ where $RSUC$ appears i times. Define $RSUC(q, 0)$ to be q. Similarly we define $LSUC(q, i)$.

Using a tree data structure based on the successor function, Chen et al. were able to compute iterated successors in constant time [4] (the same holds for the circular arcs as well).

Lemma 1. *After $O(n)$ time preprocessing, given integers $q \in [1 \ldots 2n]$ and $i \in [1 \ldots n]$ $RSUC(q, i)$ or $LSUC(q, i)$ can be computed in constant time.*

Chen et al. [4] make further use of their tree structure to attain the following.

Lemma 2. *After $O(n)$ time preprocessing, given two intervals (arcs) $I \in \mathcal{S}$ and $J \in \mathcal{S}$ ($I \in \mathcal{A}$ and $J \in \mathcal{A}$) the distance between I and J in G can be computed in constant time.*

3 1-Median

We will first consider the problem of locating one facility at a vertex (interval) of an interval graph to minimize the sum of the distances to the remaining vertices. For a candidate median interval I, $cost(I) = \sum_{J \in S} d(I, J)$.

We say an interval in $\mathcal{S}$ is maximal if it is not contained within any other interval in $\mathcal{S}$. We need not consider a non-maximal interval I_i as a candidate for a median as any interval I_j containing I_i can replace I_i as median without increasing the cost.

For a candidate median (maximal interval) I_i, the cost of servicing the other intervals can be broken down into two parts according to whether the right endpoint b_j of an interval $I_j \in \mathcal{S}$ lies left of b_i or right of b_i. Thus $Cost(I_i) = LSUM(b_i) + RSUM(b_i)$ where for an endpoint i of a maximal interval I_i we define $LSUM(i) = \sum_{I_j \in \mathcal{S} | b_j < i} d(I_j, I_i)$ and $RSUM(i) = \sum_{I_j \in \mathcal{S} | b_j > i} d(I_j, I_i)$

Then to compute the 1-median it suffices to compute $LSUM(i)$, and $RSUM(i)$ for each endpoint i of a maximal interval and let the median be the maximal interval I_k for which $LSUM(b_k) + RSUM(b_k)$ is minimum.

Let us turn to the problem of computing $LSUM(i)$ for each maximal interval endpoint i. If $\#bL(i) = 0$ then $LSUM(i) = 0$. In general, once the $LSUM$ values of all maximal interval endpoints left of i are computed, $LSUM(i)$ is computed in constant time using the formula in the following lemma.

Lemma 3. *If endpoint i is the left endpoint of a maximal interval I_i then*

$$LSUM(i) = LSUM(a_{LSUC(i)}) + \#bL(a_{LSUC(i)}) + 2 * (\#bL(i) - \#bLa_{LSUC(i)}).$$

If i is the right endpoint of maximal interval I_i then $LSUM(i) = LSUM(a_i) + \#bL(i) - \#bL(a_i)$.

Proof: To prove the first part we note that the contribution to $LSUM(i)$ of intervals whose right endpoints are left of the left endpoint of the maximal interval $LSUC(i)$ is given by $LSUM(a_{LSUC(i)}) + \#bL(a_{LSUC(i)})$. The intervals whose right endpoints lie between $a_{LSUC(i)}$ and i each contribute 2 to $LSUM(i)$.

To prove the second part, notice that the intervals whose right endpoints lie in I_i contribute 1 to $LSUM(i)$, and the contribution of other intervals is captured by $LSUM(a_i)$. ■

Similarly $RSUM(i)$ can be computed in constant time once the $RSUM$ values of maximal interval endpoints right of i have been computed. The following lemma gives the formula.

Lemma 4. *If i is the left endpoint of interval I_i then $RSUM(i) = RSUM(b_i) + \#bL(b_i) - \#bL(a_i)$. If i is the right endpoint of interval I_i then $RSUM(i) = RSUM(b_{RSUC(i)}) + \#bR(i) + \#aR(i) - \#bR(b_{RSUC(i)})$.*

Proof:

1. The sum of the distances between I_i and intervals whose right endpoints lie right of b_i is captured by $RSUM(b_i)$. The intervals whose right endpoints lie in I_i are all at distance one from I_i. There are $\#bL(b_i) - \#bL(a_i)$ such intervals.
2. Intervals which contribute to $RSUM(i)$ can be partitioned into three disjoint subsets A,B and C as follows. $I_j \in A$ if and only if $b_j > b_{RSUC(I_i)}$, $I_j \in B$ if and only if I_j contains b_i and $I_j \in C$ if and only if $b_i < a_j < b_j < b_{RSUC(I_i)}$. The definition of RSUC ensures that $I_{RSUC(i)}$ is maximal and thus A, B and C are disjoint. The contribution to $RSUM(i)$ of intervals in A is $RSUM(b_{RSUC(i)}) + \#bR(b_{RSUC(i)})$. The contribution to $RSUM(i)$ of intervals in B is $\#bR(i) - \#aR(i)$ and the contribution to $RSUM(i)$ by intervals in C is $2 * \{(\#aR(i) - \#bR(b_{RSUC(i)}))$. Adding the contributions of A, B and C yields the formula in the lemma. ■

The formulae in the previous lemmas allow the computation of $RSUM(i)$ and $LSUM(i)$ for all i such that i is an endpoint of a maximal interval in $O(n)$ time. The 1-median is the maximal interval $I_i \in \mathcal{S}$ for which $RSUM(b_i) + LSUM(b_i)$ is minimum. Thus

Theorem 1. *The 1-median of a set of intervals whose endpoints have been sorted can be computed in $O(n)$ time.*

4 *p*-median in interval graphs

In this section we consider how best to locate p facilities in intervals to minimize the total distance from the remaining intervals to their nearest facility. When locating more than one median we need to be able to quickly determine the nearest median for an interval.

If there are no more than p maximal intervals in $\mathcal{S}$ the required set of p medians will consist of the maximal intervals in $\mathcal{S}$ plus an arbitrary selection of the remaining required number of intervals. Thus in the following we assume that there are more than p maximal intervals in S and we can also again restrict our candidate medians to maximal intervals.

Consider two candidate nonadjacent medians M_1 and M_2 in a possible solution to the p-median problem, $p \geq 2$ such that $b_1 < b_2$ and no other median I_i in the candidate solution exists such that $b_1 < b_i < b_2$. For an interval I_j such that $a_1 < a_j$ and $b_j < b_2$, we want to determine whether I_j will be serviced by M_1 or M_2. How we do this depends upon the distance between M_1 and M_2.

[Even Case] First we consider the case where the distance d between M_1 and M_2 is even. See the example in the figure 1. Let p' be the left endpoint of $LSUC(M_2, \frac{d}{2} - 1)$.

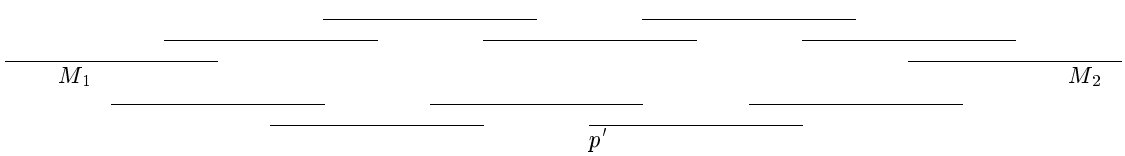

Fig. 1. The distance between M_1 and M_2 is even (six)

Claim (R,R split). Any interval with right endpoint right of p' is as close to M_2 as M_1 and any interval with right endpoint left of p' is as close to M_1 as M_2.

All intervals with right endpoint right of p' are at distance $\frac{d}{2}$ or less from M_2. They are at distance $\frac{d}{2}$ or more from M_1, or the fact that d is the minimum distance from M_1 to M_2 would be contradicted. Likewise all intervals with right endpoint left of p' are at distance at least $\frac{d}{2}+1$ from M_2 and at distance at most $\frac{d}{2}+1$ from M_1.

[Odd Case] Now we consider the case where the distance between M_1 and M_2 is odd and equal to $d+1$. See the example in figure 2. For this case let p be the right endpoint of $RSUC(M, \frac{d}{2})$. The following claim can be proven analogously to the corresponding claim in the even case.

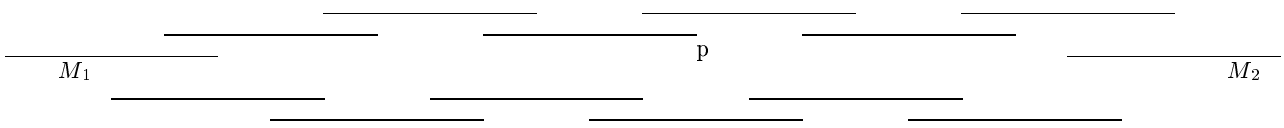

Fig. 2. The distance between M_1 and M_2 is odd (seven)

Claim (R,R split). Any interval with right endpoint left of p is as close to M_1 as M_2 and any interval with right endpoint right of p is as close to M_2 as M_1.

We have then that the set of intervals to be serviced by one median I of a group of p medians can be determined by the proximity of the right endpoints of the intervals with respect to I. In order to account for the costs of these intervals to a solution we generalize $LSUM$ and $RSUM$ as follows.

For an endpoint j of a maximal interval I_j and another maximal endpoint i such that $i < j$, define $LSUM(i,j) = \sum_{I_k \in S | i \leq b_k < j} d(I_k, I_j)$. Likewise for endpoint i of maximal interval I_i and another maximal endpoint j such that $i < j$, define $RSUM(i,j) = \sum_{I_k \in S | i < b_k \leq j} d(I_i, I_k)$.

To be able to efficiently compute $LSUM(i,j)$ and $RSUM(i,j)$ we relate these quantities to $LSUM(j)$ and $RSUM(i)$.

Lemma 5. *Let i and j, $i < j$, be endpoints of maximal intervals I_i and I_j respectively. Let I_p be the leftmost left iterated successor of I_j whose left endpoint a_p lies right of i, then*

$$\begin{aligned} LSUM(i,j) = {} & LSUM(j) - LSUM(a_p) - \#bL(a_p) * d(I_p, I_j) \\ & + (\#bL(a_p) - \#bL(i)) * (2 + d(I_p, I_j)) \end{aligned}$$

Proof: To compute $LSUM(i,j)$ the sum of the distances to I_j of the intervals whose right endpoints lie in $[i,j)$, we start with $LSUM(j)$ (the sum of the distances to I_j for intervals whose right endpoints lie left of j). Next we subtract away the contributions of intervals lying left of a_p, finally we add back in the contributions of intervals whose right endpoints lie in $[i,p)$.

The following lemma gives the analogously derived formula for $RSUM(i,j)$.

Lemma 6. *Let i and j, $i < j$, be endpoints of maximal intervals I_i and I_j respectively. Let I_p be the rightmost right iterated successor of I_i whose right endpoint b_p lies left of j, then*

$$
\begin{aligned}
RSUM(i,j) = {} & RSUM(i) - RSUM(b_p) - \#bR(b_p) * d(I_i, I_p) \\
& +(2 + d(I_i, I_p)) * \{ \textit{ The \# of intervals contained in } \\
& \qquad [p,j) = \#In[p,j]\} \\
& +(1 + d(I_i, I_p)) * \{ \textit{ The \# of intervals containing point } b_p \textit{ but not } \\
& \qquad \textit{point } j \textit{ which is } \#bL(j) - \#bL(b_p) - \#In[b_p, j]\}.
\end{aligned}
$$

All of the needed quantities can be easily computed except $\#In[p,p']$, the number of intervals contained in the interval $[p,p']$. The interval $[p,p']$ is not an interval in S and we need to be able to quickly compute the number of intervals of S contained in a query interval such as $[p,p']$. To do this we represent the intervals in S as points in the plane where interval I_i is represented by point (a_i, b_i). Then intervals in $[p,p']$ have the x coordinates of their corresponding points in the interval $[p,p']$ and the y coordinate of their corresponding points in the interval $[0,p']$. Thus the number of intervals of S contained in $[p,p']$ can be computed by range search counting query using McCreight priority search tree in $O(\log n)$ query time [16].

The previous two lemmas allow us to conclude that $LSUM(i,j)$ and $RSUM(i,j)$ can be efficiently computed.

Lemma 7. *For $i < j$ and i and j endpoints of maximal intervals, $LSUM(i,j)$ or $RSUM(i,j)$ can be computed in $O(\log n)$ time.*

With $SUM(i,j)$ and $RSUM(i,j)$ available we now turn to the dynamic programming approach of Hassin and Tamir [11] for computing medians of points on the real line.

For j the endpoint of a maximal interval in $\mathcal{S}$ and $1 \leq q \leq p$ we define $F^q(j)$ to be the size of the optimal q median solution where each of the q medians must have at least one endpoint left of or equal to j and the intervals to be serviced have right endpoints less than or equal to j.

For j the right endpoint of a maximal interval in $\mathcal{S}$ and $1 \leq q \leq p$ we define $G^q(j)$ to be the size of the solution of the same subproblem that defines $F^q(j)$ except that the interval I_j, one of whose endpoints is j, is included in the solution as one of the q medians.

Let R be the set of right endpoints of maximal intervals in $\mathcal{S}$ and let M be the set of all endpoints of maximal intervals in $\mathcal{S}$. Then for each endpoint j of a maximal interval

$$F^q(j) = min_{\{i<j \mid i \in R\}}\{G^q(i) + RSUM(i,j)\} \tag{1}$$

and for each right endpoint j of a maximal interval

$$G^q(j) = min_{\{i<j \mid i \in M\}}\{F^{q-1}(i) + LSUM(i,j)\} \tag{2}$$

with boundary conditions $F^q(1) = 0$ and $G^1(j) = LSUM(1,j)$.

Hassin and Tamir [11] exploit the quadrangle inequality in order to apply the fast dynamic programming algorithms of [6, 12] to solve recurrences similar to $F^q(j)$ and $G^q(j)$. Here we have the quadrangle inequality for $LSUM(i,j)$ and $RSUM(i,j)$.

Lemma 8. *Let j, k, l and m be endpoints of maximal intervals such that $1 \leq j \leq k \leq l \leq m$. Furthermore restrict l and m to be right endpoints of maximal intervals. Then*

$$LSUM(j,m) - LSUM(j,l) \geq LSUM(k,m) - LSUM(k,l)$$

Proof: Let $M = LSUM(j,m) - LSUM(k,m)$. Then $M = \sum_{I_t \in S | j \leq b_t < k} d(I_t, I_m)$. Let $JK = \{I_t \in S | j \leq b_t < k\}$. Then $M = \sum_{I_t \in JK} d(I_t, I_m)$. Let $L = LSUM(j,l) - LSUM(k,l)$. Then $L = \sum_{I_t \in JK} d(I_t, I_l)$. Since m and l are right endpoints of maximal intervals and $1 \leq j \leq k \leq l \leq m$, we have that $M \geq L$ and the lemma follows. ■

We can similarly show that the quadrangle inequality holds for $RSUM(i,j)$.

Lemma 9. *Let j, k, l and m be endpoints of maximal intervals such that $1 \leq j \leq k \leq l \leq m$. Furthermore restrict j and k to be right endpoints of maximal intervals. Then*

$$RSUM(j,m) - RSUM(j,l) \geq RSUM(k,m) - RSUM(k,l)$$

If $LSUM(i,j)$ and $RSUM(i,j)$ could be computed in constant time, the quadrangle inequality would allow us to follow Hassin and Tamir [11] in applying the fast dynamic programming algorithms in [6, 12] and use equations (1) and (2) to compute $G^q(j)$ and $F^q(j)$ for all $1 \leq q \leq p$ and all relevant j (j an endpoint of a maximal interval for $F^q(j)$ and j the right endpoint of a maximal interval for $G^q(j)$) in $O(pn)$ total time. However the best time we have for computing each $LSUM(i,j)$ or $RSUM(i,j)$ from lemma 4.5 is $O(logn)$. Thus we conclude

Theorem 2. *The p-median problem for a given interval graph can be computed in $O(pn \log n)$ time.*

5 *p*-median in Circular Arc Graphs

Recall that we are given a set $\mathcal{A} = \{I_1, ..., I_n\}$ of n arcs on the unit circle centered at the origin. For a given circular arc graph $G(\mathcal{A})$ we wish to find a set M of p nodes (medians) such that the sum of the distances from all nodes in $G(\mathcal{A})$ to the nearest node in M is minimized. We show how to use the algorithm for finding p medians for an interval graphs in order to solve the same problem for the circular arc graphs. Denote by T_i the running time of the algorithm that solves p-median. The main result can be stated as the following theorem.

Theorem 3. *The p-median problem for the circular arc graphs can be solved in $O(nT(n))$ time, where $T(n)$ is the running time of an algorithm that solves p-median problem for an interval graph with n nodes.*

Proof. Starting from the left endpoint of some arc $I_j \in \mathcal{A}$ we sort all the arcs in clockwise order of their left endpoints. We assume that the indices of arcs in $\mathcal{A}$ already correspond to the sorted order of arcs. As before we allow to use any integer values for indices of arcs using modulo n computation. Consider the set $M = \{I_{i_1}, I_{i_2}, \ldots, I_{i_p}\}$ of arcs that presents an optimal solution. Let C be a set of arcs with their left endpoints between i_1 and i_2 in clockwise order. Note that C is equal to $\mathcal{A}$ for $p = 1$. A set C is divided into two disjoint subsets C_1 and C_2 such that all the nearest median for all the arcs in C_1 (C_2) is I_{i_1} (I_{i_2}. In the case of the equal distance from some arc in C to I_{i_1} and I_{i_2} we associate this arc with C_1. If $p = 1$ instead of considering distances to I_{i_1} and I_{i_2} we consider distances computed in clockwise and counterclockwise directions. Obviously, the indices of the arcs in C_1 (C_2) form a consecutive sequence of integers starting at i_1 (ending at i_2). Let I_k be the arc with the largest index in C_1. We shoot a ray which passes through I_k (to the left of left endpoint of I_{k+1}) in order to unroll the circle and obtain collection of intervals, see Figure 3. Each arc from C that intersected by a shooting ray is divided into two intervals. We discard the left intervals as depicted in Figure 3 by cross. The optimal solution for an obtained interval graph corresponds to the optimal solution for $G(\mathcal{A})$. Thus, in order to solve p-median problem for a given circular arc graph $G(\mathcal{A})$ we apply n times algorithm for an interval arc graph obtained by unrolling $G(\mathcal{A})$ using a ray through every endpoint.

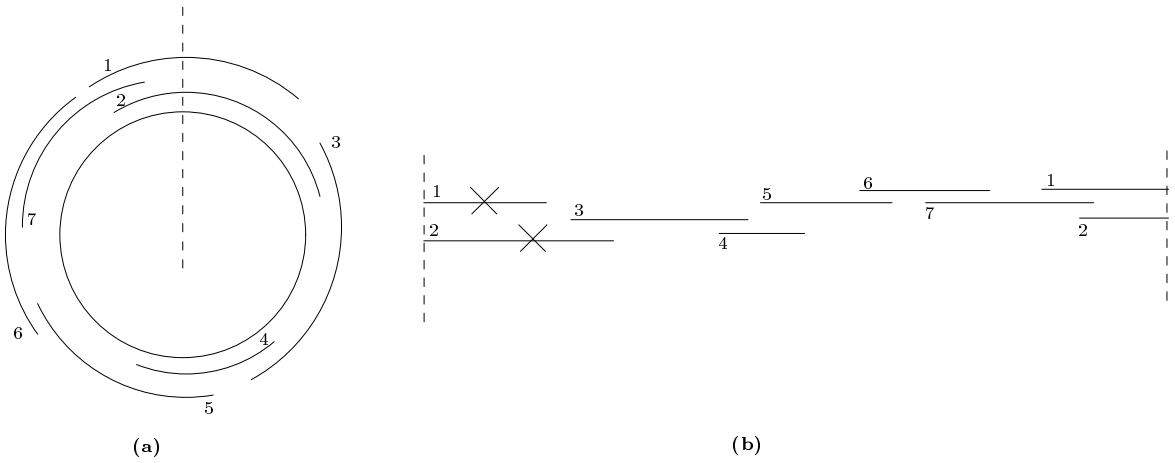

Fig. 3. (a) The initial circular arc graph; (b) unrolling circular arc graph into an interval graph, $k = 2$.

The following corollary follows immediately from the theorem.

Corollary 1. *The p-median problem for a given circular arc graph with n nodes can be solved in* $O(pn^2 \log n)$ *time.*

6 *p*-center in Circular Arc Graphs

We wish to find a set M of p nodes (centers) in $G(\mathcal{A})$ such that the maximum distance from the nodes of graph to the nearest node in M is minimized.

The algorithm below is based on *spring model* of computation. We first describe the idea of the algorithm and then provide all the details.

6.1 Spring model

Starting from the left endpoint of some arc $I_j \in \mathcal{A}$ we sort all the arcs in clockwise order of their left endpoints. We assume that the indices of arcs in $\mathcal{A}$ already correspond to the sorted order of arcs. We allow to use any integer values for indices of arcs using modulo n computation. For example, $I_1 = I_{1-n} = I_{n+1} = I_{2n+1} = \ldots$. For each pair of arcs $I_j, I_k \in \mathcal{A}$ we define a set $T_{j,k}$ of arcs such that their left endpoints are between a_j and a_k in clockwise order. This set $T_{j,k}$ can be divided into two subsets $T^1_{j,k}$ and $T^2_{j,k}$, such that the distance from every arc in $T^1_{j,k}$ ($T^2_{j,k}$) to I_j is smaller or equal (greater) than the distance to I_k. At any execution time we maintain a temporary set $C = \{I_{i_1}, \ldots, I_{i_p}\}$ of p centers such that $1 \le i_1 < i_2 < \ldots < i_p \le n$. For a sake of clarity we use C to denote centers in $G(\mathcal{A})$ and corresponding arcs as well. Initially, $C = \{I_1, \ldots, I_p\}$. For each arc $I_{i_j} \in C$ we define two *spring forces*, left force and right force as following.

- Left force $LFORCE(i_j)$ is the largest distance from arcs in $T^2_{i_{j-1},i_j}$.
- Right force $RFORCE(i_j)$ is the largest distance from arcs in $T^1_{i_j,i_{j+1}}$.

In order to obtain a new set of centers we find some center I_{i_j} in the current set C and change it to I_{i_j+1}, see Figure 4. Note, that the centers in C always move in clockwise order. However, not all the centers are allowed to move in the current moment. Let $C' \subseteq C$ be a set of centers that are allowed to move. The set C' is defined by the following rules.

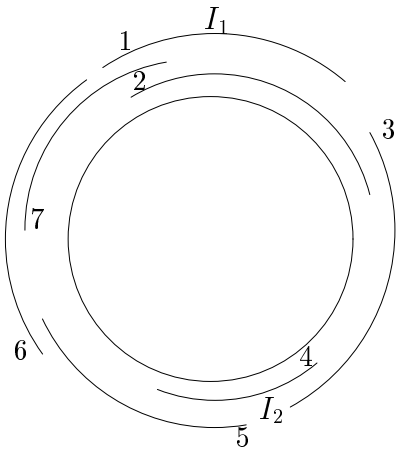

Fig. 4. For the 2-center problem the arcs I_1 and I_2 with the enpoints numbered 1 and 4, respectively, are the current centers. According to the definition $LFORCE(1) = RFORCE(1) = LFORCE(2) = 1$ and $RFORCE(2) = 2$. The arc I_2 is moved to be the arc with the endpoint numbered at 5.

Rule 1. $I_{i_j} \in C$ cannot move if $i_{j+1} = i_j + 1$.
Rule 2. If the last center I_{i_p} is equal to I_p for the second time it is not allowed to move at future.

At any time we have $2p$ computed forces. The maximal force defines the movement of centers. If the maximal force is the right force for some center then this center moves to the right. If the maximal force is the left force for some center then its left neighbor center moves to the right. Formally, the moving center I_{i_j} is defined as an arc in C' with the largest value $\max(RFORCE(i_j), LFORCE(i_{j+1}))$. and in the case of tie we can choose any of them. The algorithm stops when $C' = \emptyset$. At the end of the algorithm's execution the last set of centers C is returned.

Theorem 4. *The algorithm above correctly computes p centers for a given circular arc graph $G(\mathcal{A})$.*

Proof. Let $\{I_{c_1}, \ldots, I_{c_p}\}$ be the optimal solution. Assume, by contrary, that our algorithm misses the correct solution. Let us consider the last step of the algorithm where $C = \{I_{i_1}, \ldots, I_{i_p}\}$ and $i_1 \leq c_1, i_2 \leq c_2, \ldots, i_p \leq c_p$. The following step of algorithm makes the wrong movement of center $I_{i_j}, 1 \leq j \leq p$, which changes to I_{i_j+1}. In other words, $i_j + 1 > c_j$. It follows that $i_j = c_j$. Therefore, $T_{i_j,i_{j+1}} \subseteq T_{c_j,c_{j+1}}$ and $T^1_{i_j,i_{j+1}} \subseteq T^1_{c_j,c_{j+1}}$, $T^2_{i_j,i_{j+1}} \subseteq T^2_{c_j,c_{j+1}}$ We obtain that $\max(RFORCE(i_j), LFORCE(i_{j+1})) \leq \max(RFORCE(c_j), LFORCE(c_{j+1}))$ and, thus, a set C presents an optimal solution.

6.2 Implementation

In order to find forces efficiently we use the data structure proposed by Chen et al. [4]. This data structure can be constructed in $O(n)$ time and $O(n)$ space. Using this data structure the query on the length of the shortest path between any two arcs can be answered in $O(1)$ time. This data structure also supports computing iterated right and left successors for a given integer value in constant time. We use Lemma 1 and Lemma 2 which provide an efficient way to maintain spring model.

Proof. We distinguish between two cases : even and odd distance d. If d is odd then there is some arc I_k at distance $\frac{d-1}{2}$ and $\frac{d+1}{2}$ from centers I_{i_j} and $I_{i_{j+1}}$. See Figure 5.

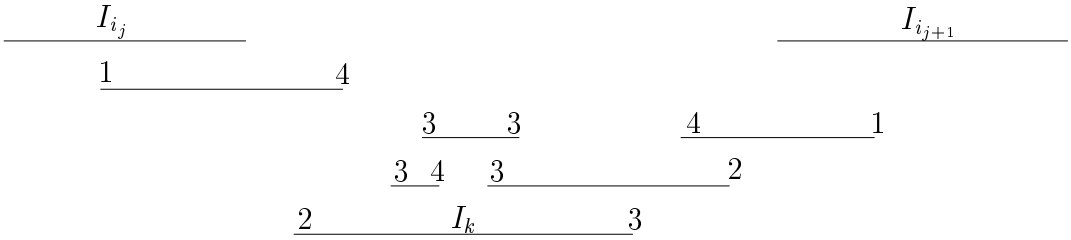

Fig. 5. The numbers on the arcs denote the distances to corresponding centers.

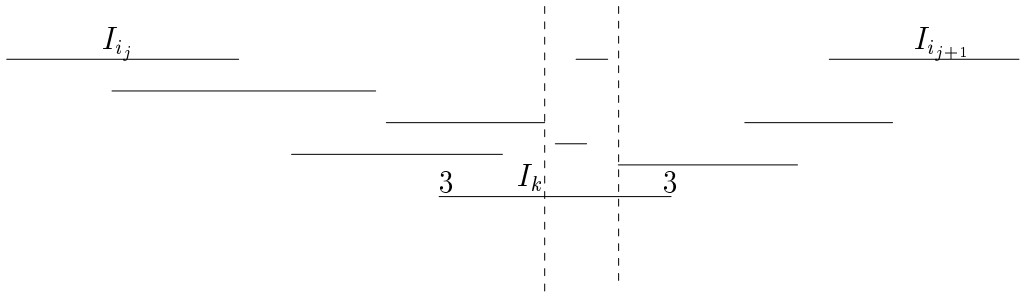

Fig. 6. The dashed lines show the slab.

We claim that the maximum force is equal to $\frac{d+1}{2}$, i.e. there is no arc at distance greater than $\lceil \frac{d}{2} \rceil$ from both I_{i_j} and $I_{i_{j+1}}$. Assume, by contrary, that such arc I_l exists. If arcs I_k and I_l intersect then distance of I_l to one of the centers is at most $\frac{d-1}{2} + 1 = \frac{d+1}{2}$. Otherwise, the arc I_l is either to the left of I_k or to the right. The distance from I_l to the corresponding center is less or equal than the distance from I_k to the same center. It contradicts the definition of I_l.

It remains to show how to deal with the even case. See Figure 6. The arc I_k is at distance $\frac{d}{2}$ from both centers I_{i_j} and $I_{i_{j+1}}$. We define a vertical slab for I_k whose left side passes through the right endpoint of arc $RSUC(i_j, d/2 - 1)$ and right side passes though the left endpoint of arc $LSUC(i_{j+1}, d/2 - 1)$. We claim that the maximum force is equal to $d/2$ if and only if this slab contains no arcs. Otherwise the maximum force equals $d/2 + 1$. Note that all the arcs containing some point to the left of a slab (including left side of a slab) are at distance at most $d/2$ from I_{i_j}. Similarly, all the arcs containing some point to the right of a slab (including right side of a slab) are at distance at most $d/2$ from $I_{i_{j+1}}$. Thus, if slab does not contain any arc clearly all arcs are within a distance $d/2$ from centers; otherwise the arcs inside of a slab are at distance $d/2 + 1$ from each center.

The computation of length between two arcs as well as computation of the right and left iterated successors can be done in constant time [4]. Determining whether a slab contains any arc can be done also in constant time using array T whose indices corresponds to the left endpoints of the arcs and for any given index $i, 1 \leq i \leq n$, $T[i]$ keeps the rightmost left endpoint of the arcs with right endpoint less than i. This array T can be precomputed in linear time after once the endpoints of the arcs are sorted. Slab contains some arc if and only if the value of element of T corresponding to the right side of slab is larger than the left side of the slab.

We conclude by theorem.

Theorem 5. *The algorithm computes p centers for a given circular arc graph $G(\mathcal{A})$ in $O(np)$ time if the endpoints of the arcs are sorted.*

Remark. Applying the spring model strategy to the p-median problem in the circular-arc graph leads to the similar result obtained in Section 5.

Acknowledgements

The authors acknowledge the financial support received from the Natural Sciences and Engineering Research Council of Canada.

References

1. J. Bar-Ilan and D. Peleg, "Approximation algorithms for selecting network centers", In *Proc. Workshop on Algorithms and Data Structures'91*, 1991, 343–354.
2. K.S. Booth and G.S. Leuker, "Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms", *Journal of Computer and System Sciences*, 13 (1976), 335 – 379.
3. M.L. Brandeau and S.S. Chiu, "An overview of representative problems in location research" *Management Science*, 35 (1989), 645-674.
4. D. Chen , D.T. Lee, R. Sridhar and C. Sekharam, "Solving the All-Pair Shortest Path Query Problem on Interval and Circular-Arc Graphs", *Networks*, to appear.
5. G. Frederickson, "Parametric search and locating supply centers in trees", In *Proc. Workshop on Algorithms and Data Structures'91*, 1991, 299–319.
6. Z. Galil and K. Park, "A linear-time algorithm for concave one-dimensional dynamic programming", *Information Processing Letters*, 33 (1990), 309–311.
7. B. Gavish and S. Sridhar, "Computing the 2-Median on Tree Networks in $O(n \log n)$ Time", *Networks*, 26 (1995), 305 – 317.
8. A. J. Goldman, "Optimal center location in simple networks", *Transportation Science*, 5 (1971), 212 – 221.
9. M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
10. S. L. Hakimi, "Optimal locations of switching centers and the absolute centers and medians of a graph", *Operations Research*, 12 (1964), 450 – 459.
11. R. Hassin and A. Tamir, "Improved complexity bounds for location problems on the real line", *Operations Research Letters*, 10 (1991), 395–402.
12. M. Klawe, "A simple linear time algorithm for concave one-dimensional dynamic programming", Technical Report 89-16, University of British Columbia, Vancouver, 1989.
13. O. Kariv and S. L. Hakimi, "An algorithmic approach to network location problems I: The p-centers", *SIAM Journal on Applied Mathematics*, 37 (1979), 514 – 538.
14. O. Kariv and S. L. Hakimi, "An algorithmic approach to network location problems II: The p-medians", *SIAM Journal on Applied Mathematics*, 37 (1979), 539 – 560.
15. M. Labbe, D. Peeters and J.F. Thisse, "Location on Networks", in *Handbooks in Operations Research and Management Science*, 8, Ball et al editors, Elsevier Science, 1995.
16. E. M. McCreight, "Priority Search Trees", *SIAM Journal on Computing*, 14 (1985), 257 – 276.
17. P. Mirchandani, "The p-median problem and generalizations", in *Discrete Location Theory*, Mirchandani and Francis editors, Wiley, 1990, 55 – 117.
18. R. Mohring, "Graph problems related to gate matrix layout and PLA folding", *Computational Graph Theory*, Tinhofer et al editors, Springer-Verlag, 1990, 17 – 51.
19. S. Olariu, "A simple linear-time algorithm for computing the center of an interval graph", *International Journal of Computer Mathematics*, 24 (1990), 121-128.
20. A. Tamir, "An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs", *Operations Research Letters*, 19 (1996), 59 – 64.
21. B.C. Tansel, R. L. Francis and T. J. Lowe, "Location on Networks: A Survey - Part I: The p-center and p-median problems", *Management Science*, 29 (1983), 482 – 497.