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

Sergei Bespamyatnikh† Binay Bhattacharya‡ Mark Keil§

David Kirkpatrick¶ Michael Segal‖

Abstract

The p-center problem is to locate p facilities on a network so as to minimize the largest distance from a demand point to its nearest facility. The p-median problem is to locate p facilities on a network so as to minimize the average distance from a demand point to its closest facility. We consider these problems when the network can be modelled by an interval or circular-arc graph whose edges have unit lengths. 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 a circular-arc graph in $O(pn^2 \log n)$ time. We introduce a spring representation of the objective function and show how to solve the p-center problem on a circular-arc graph in $O(pn)$ time, assuming that the arc endpoints are sorted.

Keywords: Interval and circular-arc graphs, p-medians, p-centers.

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 have been researched extensively [4, 10, 14, 21, 24, 26, 31, 32]. Applications include the location of industrial

*This work has been supported by a NSERC grant. Work by Michael Segal has been also supported by the Pacific Institute for Mathematical Studies.

†Dept. of Computer Science, Duke University, Box 90129, Durham, NC 27708, USA. E-mail: besp@cs.duke.edu

‡School of Computing Science, Simon Fraser University, Burnaby, B.C. Canada, V5A 1S6. Email: binay@cs.sfu.ca

§Dept. of Computer Science, University of Saskatchewan, 57 Campus Dr., Saskatoon, Sask., Canada, S7N 5A9. Email: keil@cs.usask.ca

¶Dept. of Computer Science, University of British Columbia, Vancouver, B.C. Canada V6T 1Z4. Email: kirk@cs.ubc.ca

‖Communication Systems Engineering Department, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel. E-mail: segal@cse.bgu.ac.il

plants, warehouses, and public service facilities on transportation networks as well as the location of various service facilities in telecommunication networks [4, 14, 24, 26, 32].

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 [14] 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. We assume that the edges of E have unit lengths. Let $d(u, v)$ be the minimum number of edges in the shortest path from between vertices u and v. We denote $d(u, U) = \min_{v \in U} d(u, v)$ for a set $U \subset V$. 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 [14] optimally solved it in a general network in $O(n^3)$ time. In a tree network Goldman [11] and Kariv and Hakimi [21] 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 [10]. Auletta et al. [1] consider the dynamic p-median problem on a tree where $p = 1, 2$.

For general p, Kariv and Hakimi [21] 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 [21]. The tree network algorithm was recently improved to $O(pn^2)$ by Tamir [31]. Hsu et al. [18] presented a general p-facility location problem on the line with unimodal distance functions. For this specially structured problem, they gave an $O(pn^2)$ algorithm. It is easy to see that the p-median and p-center problems on general interval graphs (with unit edge lengths), are special cases of the model in [18], and therefore they can be solved in $O(pn^2)$ time. In this paper we improve this complexity bound, and give an $O(pn \log n)$ algorithm for the p-median problem on interval graphs. Since the p-median problem on circular-arc graphs can be decomposed into n problems on interval graphs, we obtain $O(pn^2 \log n)$ time algorithm for the former class of graphs. To the best of our knowledge, other than for trees, no polynomial time algorithm for the p-median has been explicitly reported in literature for any large class of graphs. We observe, though, that for series-parallel graphs, and even for k-trees, k fixed, polynomial algorithms for the related uncapacitated facility location problem, can be easily be modified to obtain polynomial time algorithms for the p-median problem (see, for example, [15, 13]). We also note that using the technique of totally balanced matrices (see Section 6.4 in [23]), one can obtain polynomial time algorithms for some classes of intersection graphs with unit edge lengths, e.g., interval graphs and strongly chordal graphs. The running time of these algorithms, however, is expressed by high-degree polynomials.

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 an $O(n)$ time algorithm for locating a single facility in interval graph that minimizes the maximum distance to a demand point [28]. Frederickson [8] showed how to solve this problem for trees in optimal linear time (not necessarily restricting the facilities to be the vertices of the tree) using parametric search. The work of Kariv and Hakimi [20] presents results for general graphs; however since the problem is known to be NP-complete they were able to give only an $O(n^{2p+1} \log n/(p-1)!)$ runtime algorithm. Tamir [30] has shown that the weighted and unweighted p-center on networks can be found in $O(n^p m^p \log^2 n)$ and $O(n^{p-1} m^p \log^3 n)$ time, respectively, where m is the number of edges. Some work also has been done for approximating the p-center and p-median solution, see e.g. [2, 5, 6, 17, 19].

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 [12, 27].

The problem of recognizing interval and circular-arc graphs is known to be solvable in $O(|V| + |E|)$ time (see e.g, [3]), 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 graphs. 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 $O(pn)$ 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 $j \in [1 \dots 2n]$ 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 to the 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 $q \in [1 \dots 2n]$. 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 [7] (see also [29]) 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 for an integer q, its ith iterated right successor $RSUC(q,i)$ to be $RSUC(RSUC(\ldots RSUC(q,0)))$ where $RSUC$ appears i times. Define $RSUC(q,0)$ to be q. Similarly we define $LSUC(q,i)$. For any interval I_j, we define $LSUC(I_j,i)$ to be $LSUC(a_j,i)$ and $RSUC(I_j,i)$ to be $RSUC(b_j,i)$.

Using a tree data structure based on the successor function, Chen et al. were able to compute iterated successors in constant time [7] (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. [7] 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 $RSUM(i) = \sum_{I_j \in \mathcal{S} | b_j > i} d(I_j, I_i)$ Here i in $LSUM(i)$ or $RSUM(i)$ could be a_i or b_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.

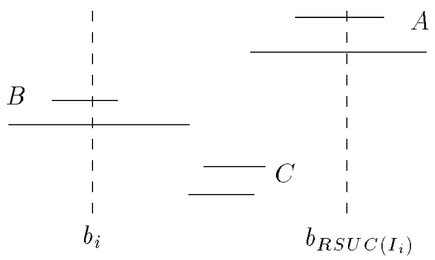

Figure 1: Sets A, B, and C.

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* $LSUM(i) = LSUM(a_{LSUC(i)}) + \#bL(a_{LSUC(i)}) + 2 * (\#bL(i) - \#bL(a_{LSUC(i)}))$. *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 5 *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. For simplicity we assume that the interval graph is connected.

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 non-adjacent 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.

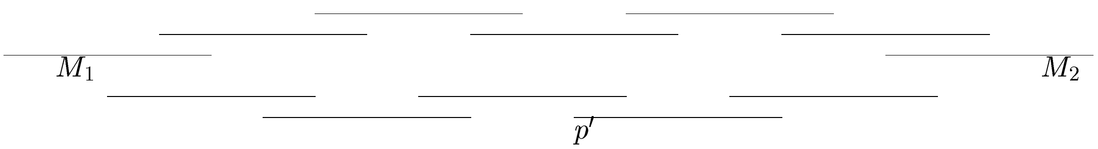

Figure 2: The distance between M_1 and M_2 is even (six).

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

Claim 6 (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.

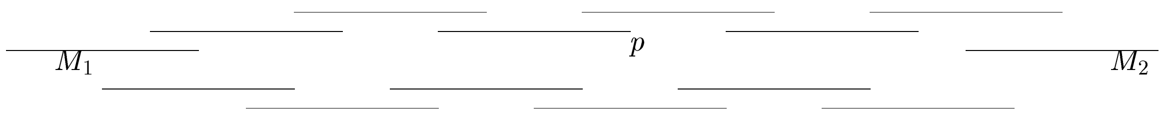

Figure 3: The distance between M_1 and M_2 is odd (seven).

[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.

Claim 7 (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-interval 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-interval 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 8 *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 $[i,a_p)$. ∎

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

Lemma 9 *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*

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

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-coordinates of their corresponding points in the interval $[p,p']$. Thus the number of intervals of S contained in $[p,p']$ can be computed by range search counting query, using the priority search tree of McCreight in $O(\log n)$ query time [25].

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

Lemma 10 *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 $LSUM(i,j)$ and $RSUM(i,j)$ available we now turn to the dynamic programming approach of Hassin and Tamir [16] for computing medians of points on the real line.

For j the endpoint of a maximal interval in $\mathcal{S}$ and $1 \le q \le p$ 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 $1 \le q \le p$ 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 [16] exploit the quadrangle inequality in order to apply the fast dynamic programming algorithms of [9, 22] 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 11 *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 12 *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 [16] in applying the fast dynamic programming algorithms in [9, 22] 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 is an endpoint of a maximal interval for $F^q(j)$ and j is 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 10 is $O(\log n)$. Thus we conclude

Theorem 13 *The p-median problem for a given interval graph can be computed in $O(pn \log n)$ time.*

Remark. One can observe that omitting the assumption about the graph connectivity does not affect the running time of the algorithm from Theorem 13. This follows from the following observation. If the interval graph has r connected components, the j-median problem for $j = 1, 2, \ldots, p$ in the ith component can be solved in $O(pn_i \log n_i)$ time where n_i is the number of intervals in the ith component. Therefore, the j-median problem for $j = 1, 2, \ldots p$ in all the connected components can be solved in $O(pn \log n)$ time.

Once the solutions to each individual component are known, it is possible to merge all the solutions in $O(p^2 \log r)$ time using the divide and conquer technique to determine the p-median solution of the interval graph. It is clear that p-median solution does not exist if $r > p$.

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 the 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 their nearest nodes in M is minimized. We show how to use the algorithm for finding p medians for an interval graph in order to solve the same problem for the circular arc graphs.

The main result can be stated as the following theorem.

Theorem 14 *The p-median problem for a circular arc graph with n nodes can be solved in $O(nT(n))$ time, where $T(n)$ is the running time of an algorithm that solves the 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 the clockwise order of their left endpoints. We assume that the indices of arcs in $\mathcal{A}$ already correspond to the sorted order of the arcs. We allow the use of any integer values for indices of arcs, using modulo n computation. For example, $I_1 = I_{1-n} = I_{n+1} = I_{2n+1} = \ldots$. Consider the set $M = \{I_{i_1}, I_{i_2}, \ldots, I_{i_p}\}$ of arcs that represent an optimal solution. Let C be the set of arcs whose left endpoints are between i_1 and i_2 in the clockwise order. Note that C is equal to $\mathcal{A}$ for $p = 1$. The set C is divided into two disjoint subsets C_1 and C_2 such that the nearest median for all arcs in C_1 (C_2) is I_{i_1} (I_{i_2}). In the case when some arc in C has the same distances to I_{i_1} and I_{i_2}, we put this arc into C_1. If $p = 1$ then instead of considering distances to I_{i_1} and I_{i_2} we consider distances computed in the 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 the intervals, see Figure 4. Each arc from C that is intersected by a shooting ray is divided into two intervals. We discard the left intervals as depicted in Figure 4.

The optimal solution for the resulted interval graph corresponds to the optimal solution for $G(\mathcal{A})$. Thus, in order to solve the p-median problem for a given circular arc graph $G(\mathcal{A})$ we apply n times the algorithm for an interval arc graph obtained by unrolling $G(\mathcal{A})$ using a ray through every endpoint. ∎

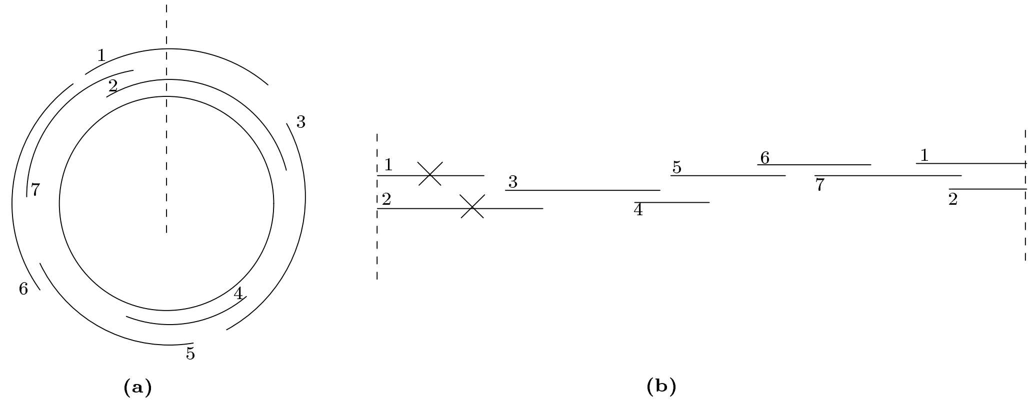

Figure 4: (a) The initial circular arc graph; (b) Unrolling the circular arc graph into an interval graph, $k = 2$; discarded intervals are marked by a cross.

The following corollary follows immediately from the theorem.

Corollary 15 *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 the graph $G(\mathcal{A})$ to the nearest node in M is minimized.

The algorithm below is based on a *spring representation* of the objective function. We first describe the idea of the algorithm and then provide technical details.

6.1 Spring model

Starting from the left endpoint of some arc $I_j \in \mathcal{A}$ we sort all the arcs in the 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 the use of any integer values for indices of arcs, using modulo n computation. For each pair of arcs $I_j, I_k \in \mathcal{A}$ we define the set $T_{j,k}$ of arcs such that their left endpoints are between a_j and a_k in the clockwise order. The 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 (greater) than or equal to 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 $1 \leq i_1 < i_2 < \ldots < i_p \leq n$. 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 defined as the largest distance between arcs in $T^2_{i_{j-1}, i_j}$ and I_{i_j}.
- Right force $RFORCE(i_j)$ is defined as the largest distance between arcs in $T^1_{i_j, i_{j+1}}$ and I_{i_j}.

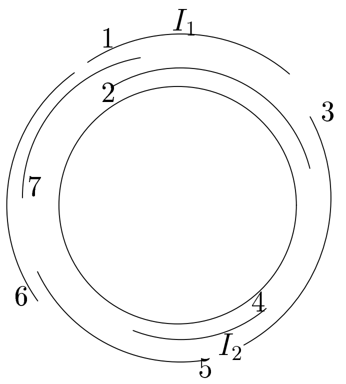

Figure 5: For the 2-center problem the arcs I_1 and I_2 with the endpoints numbered 1 and 4, respectively, are the current centers. According to the definition $LFORCE(1) = RFORCE(1) = LFORCE(2) = 1$ and $RFORCE(2) = 2$. The arc I_2 is moved to be the arc with the endpoint numbered at 5.

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 5. Note, that the centers in C always move in the 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.

Rule 1. $I_{i_j} \in C$ cannot move if $i_{j+1} = i_j + 1$, i.e. $I_{i_j} \notin C'$.

Rule 2. If the last arc in C is I_n then it cannot move.

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 the left neighbor of this center moves to the right. Formally, the moving center I_{i_j} is defined as an arc in C' with the largest value $\max_{I_{i_j} \in C'} (RFORCE(i_j), LFORCE(i_{j+1}))$, and in the case of tie we can choose any of them. The algorithm stops when $C' = \emptyset$. The rule 2 prevents infinite cycling. The rule 1 preserves the order of the current centers and their number. At the end of the algorithm's execution the best set of centers computed in the course of the algorithm. In other words, the set of p centers C' minimizing the maximum force over all sets C' is the optimal p-center.

Theorem 16 *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, to the 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 next step of the 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 $i_j = c_j$.

Let d_1 be the distance between the arcs I_{i_j} and $I_{i_{j+1}}$. Let d_2 be the distance between the arcs I_{c_j} and $I_{c_{j+1}}$. We have $d_1 \leq d_2$ since $i_j = c_j$ and $i_{j+1} \leq c_{j+1}$. The result in Section 6.2 implies that

$$\max\left(RFORCE(i_j), LFORCE(i_{j+1})\right) = \left\lceil \frac{d_1}{2} \right\rceil \quad \text{and}$$
$$\max\left(RFORCE(c_j), LFORCE(c_{j+1})\right) = \left\lceil \frac{d_2}{2} \right\rceil .$$

Therefore, $\max\left(RFORCE(i_j), LFORCE(i_{j+1})\right) \leq \max\left(RFORCE(c_j), LFORCE(c_{j+1})\right)$ and, thus, the 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. [7]. Their 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 the spring model.

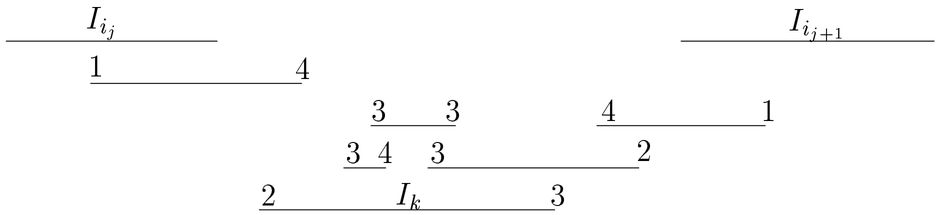

Figure 6: Odd case in Theorem 17. The numbers on the arcs denote the distances to corresponding centers.

Let d be the distance between I_{i_j} and $I_{i_{j+1}}$. We distinguish between two cases : even and odd distance d. If d is odd then there is an arc I_k at distance $\frac{d-1}{2}$ and $\frac{d+1}{2}$ from the centers I_{i_j} and $I_{i_{j+1}}$. See Figure 6. 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, to the contrary, that such an arc I_l exists. If arcs I_k and I_l intersect then the 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 to the distance from I_k to the same center. This contradicts the definition of I_l.

It remains to show how to deal with the even case. See Figure 7. Let I_k be the arc 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 whose 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 the slab (including the left side) are at distance at most $d/2$ from I_{i_j}. Similarly, all the arcs containing a point to the right of the slab (including the right side) are at distance at most $d/2$ from $I_{i_{j+1}}$. Thus, if the slab does not contain any arc, clearly all arcs are within the distance $d/2$ from centers; otherwise the arcs inside the slab are at distance $d/2+1$ from each center.

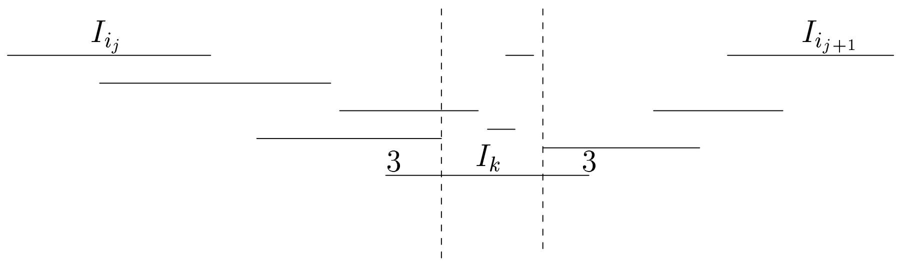

Figure 7: Even case in Theorem 17. The dashed lines show the slab.

The computation of the length between two arcs as well as the computation of the right and left iterated successors can be done in constant time [7]. Determining whether the slab contains any arc also can be done in constant time. We define the 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. The 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 the following theorem.

Theorem 17 *The algorithm computes p centers for a given circular arc graph $G(\mathcal{A})$ in $O(pn)$ time if the endpoints of the arcs are sorted.*

References

[1] V. Auletta, D. Parente and G. Persiano, "Dynamic and static algorithms for the optimal placement of resources in a tree", *Theoretical Computer Science*, 165 (1996), 441–461.

[2] J. Bar-Ilan and D. Peleg, "Approximation algorithms for selecting network centers", In *Proc. Workshop on Algorithms and Data Structures'91*, 1991, 343–354.

[3] 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.

[4] M.L. Brandeau and S.S. Chiu, "An overview of representative problems in location research" *Management Science*, 35 (1989), 645-674.

[5] M. Charikar and S. Guha, "Improved combinatorial algorithms for the facility location and k-median problems", In *Proc. 40th Annual Symposium on Foundations of Computer Science*, 1999, 378 – 388.

[6] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys, "A constant factor approximation for the k-median problem", In *Proc. 31th Annual ACM Symposium on Theory of Computing*, 1999, 1 – 10.

[7] 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*, 31(4), 1998, 249–258.

[8] G. Frederickson, "Parametric search and locating supply centers in trees", In *Proc. Workshop on Algorithms and Data Structures'91*, 1991, 299–319.

[9] Z. Galil and K. Park, "A linear-time algorithm for concave one-dimensional dynamic programming", *Information Processing Letters*, 33 (1990), 309–311.

[10] B. Gavish and S. Sridhar, "Computing the 2-median on tree networks in $O(n \log n)$ time", *Networks*, 26 (1995), 305 – 317.

[11] A. J. Goldman, "Optimal center location in simple networks", *Transportation Science*, 5 (1971), 212 – 221.

[12] M.C. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, New York, 1980.

[13] D. Granot and D. Skorin-Kapov, "On some optimization problems on k-trees and partial k-trees", *Discrete Applied Mathematics*, 48 (1994), 129 – 145.

[14] S. L. Hakimi, "Optimal locations of switching centers and the absolute centers and medians of a graph", *Operations Research*, 12 (1964), 450 – 459.

[15] R. Hassin and A. Tamir, "Efficient algorithms for optimization and selection in series-parallel graphs", *SIAM J. Algebraic Discrete Methods*, 7 (1986), 379–389.

[16] R. Hassin and A. Tamir, "Improved complexity bounds for location problems on the real line", *Operations Research Letters*, 10 (1991), 395–402.

[17] D. Hochbaum and D. B. Shmoys, "A unified approach to approximation algorithms for bottle neck problems", *JACM*, 33:3 (1986), 533–550.

[18] V.N. Hsu, T.J. Lowe and A. Tamir, "Structured p-facility location problems on the line solvable in polynomial time", *Operations Research Letters*, 21 (1997), 159 – 164.

[19] K. Jain and V. V. Vazirani, "Primal-dual approximation algorithms for metric facility location and k-median problems", In *Proc. 40th Annual Symposium on Foundations of Computer Science*, 1999, 2 – 13.

[20] 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.

[21] 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.

[22] M. Klawe, "A simple linear time algorithm for concave one-dimensional dynamic programming", Technical Report 89-16, University of British Columbia, Vancouver, 1989.

[23] A. Kolen and A. Tamir, "Covering problems", In P. Mirchandani and R. Francis (Eds.), *Discrete Location Theory*, Wiley, 1990, 263–304.

[24] 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.

[25] E. M. McCreight, "Priority search trees", *SIAM Journal on Computing*, 14 (1985), 257 – 276.

[26] P. Mirchandani, "The p-median problem and generalizations", in *Discrete Location Theory*, Mirchandani and Francis editors, Wiley, 1990, 55 – 117.

[27] R. Mohring, "Graph problems related to gate matrix layout and PLA folding", *Computational Graph Theory*, Tinhofer et al editors, Springer-Verlag, 1990, 17 – 51.

[28] S. Olariu, "A simple linear-time algorithm for computing the center of an interval graph", *International Journal of Computer Mathematics*, 24 (1990), 121-128.

[29] R. Ravi, M. Marathe and C. Rangan, "An optimal algorithm to solve the all-pair shortest path problem on interval graphs", *Networks*, 22 (1992), 21–35.

[30] A. Tamir, "Improved complexity bounds for center location problems on networks by using dynamic data structures", *SIAM J. Discrete Math.*, 1 (1988), 377–396.

[31] A. Tamir, "An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs", *Operations Research Letters*, 19 (1996), 59 – 64.

[32] 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.