k Best Cuts for Circular-Arc Graphs*

K. H. Tsai
Institute of Information Science
Academia Sinica
Nankang, Taiwan
e-mail: tsaikh@iis.sinica.edu.tw

D. T. Lee
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, Illinois 60208
e-mail: dtlee@eecs.nwu.edu

Abstract

Given a set of n nonnegative *weighted* circular-arcs on a unit circle, and an integer k, the k Best Cuts for Circular-Arcs problem, abbreviated as k-BCCA problem, is to find a placement of k points, called *cuts*, on the circle such that the total weight of the arcs that contain at least one cut is maximized.

We first solve a simpler version using dynamic programming, the k Best Cuts for Intervals (k-BCI) problem in $O(kn + n \log n)$ time and $O(kn)$ space. The algorithm is then extended to solve a variation, called the k-restricted BCI problem, and the space complexity of the k-BCI problem can be improved to $O(n)$. Based on these results, we then show that the k-BCCA problem can be solved in $O(I(k, n) + n \log n)$ time, where $I(k, n)$ is the time complexity of the k-BCI problem. As a by-product, the k Maximum Cliques Cover problem, ($k > 1$) for the circular-arc graphs can be solved in $O(I(k, n) + n \log n)$ time.

*Supported in part by the National Science Foundation under the Grants CCR-8901815, CCR-9309743 and INT-9207212, and by the Office of Naval Research under the Grant No. N00014-93-1-0272.

1 Introduction

We address problems in competitive location theory [7] in the plane, the general setting of which is: given a set of demand points in the plane and a set of existing facilities serving the demand points, we want to locate a set of new facilities so that they will *compete* favorably against the existing facilities under some distance criteria. Drezner [5] studied the following *one-on-one competition problem*: given a set of demand points p_i with associated weight $w_i, i = 1, \ldots, n$ in the plane and an existing facility X, locate a new facility Y so that Y will attract most total weight; a demand point p_i (and its associated weight w_i) is considered attracted to Y if $d(p_i, Y) < d(p_i, X)$, where $d(a, b)$ is the Euclidean distance between a and b. He gave an $O(n \log n)$ algorithm, and was later proven by Lee and Wu [11] to be optimal under the algebraic computation tree model of Ben-Or [4].

The Minimum New Facilities (MNF) problem is: given an existing facility X serving n demand points $p_i = (x_i, y_i)$, $i = 1, \ldots, n$, find a minimum number m of new facilities at least R away from X so as to attract maximum number of demand points; the competition rule is similar to that of the one-on-one competition problem.

Lee, Sarrafzadeh, and Wu[10] transformed the MNF problem to the Minimum Cuts for Circular-Arcs problem as follows.

Let's assume that the existing facility is at the origin O. Denote the circle of radius R centered at the origin as the R-circle. It is shown that it suffices to consider facilities located on the R-circle only. For each demand point p_i, let R_i be the circle centered at p_i and of radius r_i, which equals $d(p_i, O)$. If there is a new facility Y inside the circle R_i, then p_i is attracted by Y. If $r_i \leq R/2$, then R_i is contained in the R-circle, and it is impossible to attract p_i away from the existing facility at the origin with any facility at least R away from the origin.

Let A_i be the portion of R-circle inside the circle R_i. Then, A_i is a circular-arc, and can be represented as $(\theta_{ib}, \theta_{ie})$, where θ_{ib} (θ_{ie}, resp.) is the angle of clockwise (counterclockwise) end of A_i. A_i is an open arc on the R-circle, and called a *demand arc* with the following implication: if there is any facility located on A_i, then p_i is attracted. (Figure 1.)

Note that the values of the angles of demand arcs are taken modulo 2π. We obtain the following problem.

Minimum Cuts for Circular-Arcs (MCCA) Problem: Given n nonempty open arcs $A_i = (\theta_{ib}, \theta_{ie})$, $i = 1, \ldots, n$ on the unit circle, where $\theta_{ib} \in [0, 2\pi)$ is the beginning angle (the clockwise end, or the CW end for short) and $\theta_{ie} \in [0, 2\pi)$ is the ending angle (the counterclockwise end, or the CCW end for short) of each arc A_i, and the length of each arc A_i is less than π, place a minimum number m of cuts at angles $\alpha_1, \ldots, \alpha_m$ such that every arc intersects at least one cut. A cut at angle α denotes a half-line emanating from the origin with orientation α.

It has been shown [10] that the MCCA (and hence the MNF) problem can be solved in $\Theta(n \log n)$ optimal time.

In this paper we consider an extension of the MNF problem, which was first considered by Wu[13].

k **Best New Facilities (k-BNF) Problem**: Given an existing facility X serving n demand points $p_i = (x_i, y_i)$, with positive weight w_i $i = 1, \ldots, n$, and $k \leq m$, where m is the answer to the MNF problem, locate k new facilities at least R away from X so as to attract most total weight.

Using the same transformation from MNF to MCCA, we transform the k-BNF problem to the following problem.

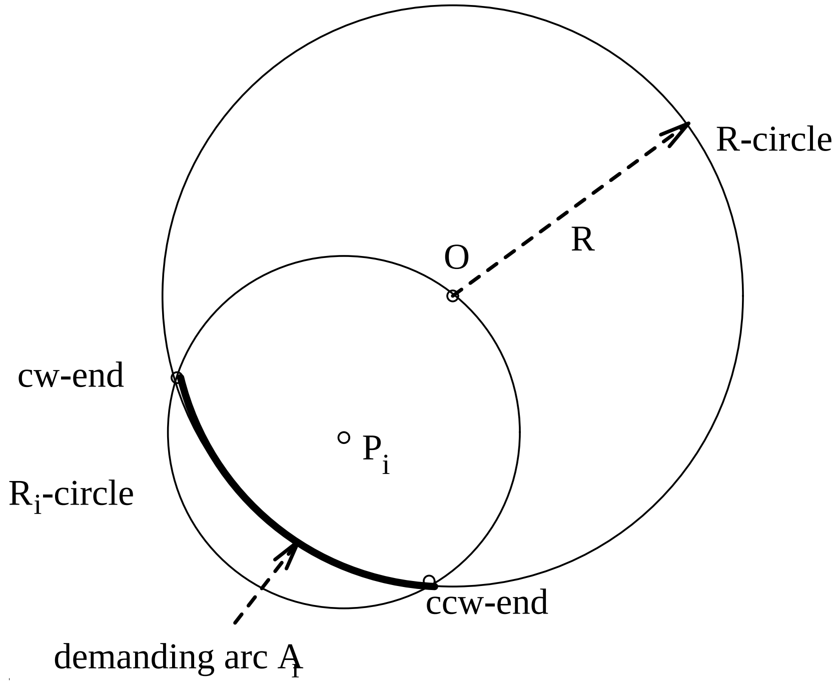

Figure 1: Transformation from a demand point to a demand arc

k **Best Cuts for Circular Arcs (k-BCCA) Problem**: Given n nonempty open arcs A_i with positive weight w_i, $i = 1, \ldots, n$, each having length less than π, on the unit circle, and $k \le m$, where m is the number of cuts in an optimal placement of cuts that intersect all arcs, find a set of k cuts so as to maximize the total weight of those arcs intersected by at least one cut.

Note that when $k = m$, the problem is exactly the MCCA problem.

Definition 1 *A cut interval is a nonempty angular interval $c = (\theta_b, \theta_e)$ or $[\theta_b, \theta_e)$ such that $\theta_e \in \{\theta_{ie} | \ i = 1, \ldots, n\}$ and $\theta_b \in \{\theta_{ib} | \ i = 1, \ldots, n\} \cup \{\theta_{ie} | \ i = 1, \ldots, n\}$, and for all $\theta_1, \theta_2 \in c$, the set of arcs intersecting the cut at θ_1 is the same as the set of arcs intersecting the cut at θ_2; c does not contain θ_b, i.e., open on its CW end, iff $\theta_b \in \{\theta_{ib} | \ i = 1, \ldots, n\}$.*

In general, each θ_{ie} determines a cut interval; thus we have n cut intervals unless some of the θ_{ie}'s of the n given arcs are not distinct. Let us denote the set of $t \le n$ cut intervals as $C = \{c_1, \ldots, c_t\}$. We shall use the phrase "arc A_i is *covered* by cut interval c_j" to mean that any cut located within c_j intersects A_i. Figure 2 shows an example with arcs $A_1, \ldots, A_8$, and cut intervals $c_1, \ldots, c_7$; e.g., c_1 is open on both ends, while c_7 is closed on the CW end and open on the CCW end.

The following fact is obvious.

Fact 1: It is sufficient to place at most one cut inside each cut interval and none outside the union of all cut intervals.

In the next section we shall consider a simpler version of k-BCCA problem, the interval model, called k-BCI problem, where the input is a set of intervals, instead of circular-arcs. In section 3 we consider k-RBCI problem, in which we are givin, in addition to the input intervals, k intervals where each of the k cuts should be restricted. Some properties on the best k cuts are introduced in Section 4. In Section 5, using algorithms for these two problems as subroutines we present an

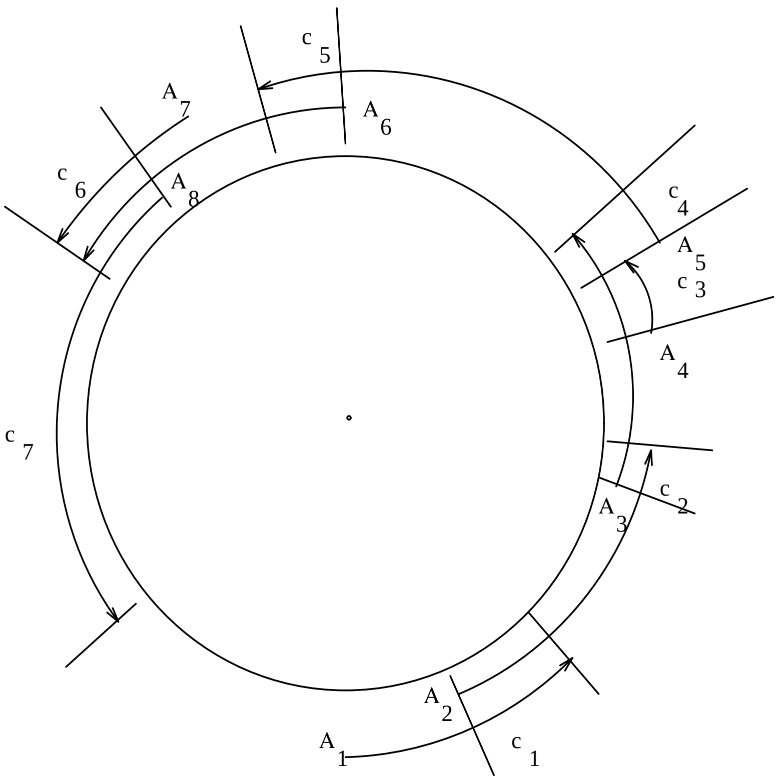

Figure 2: Cut intervals.

algorithm for solving the k-BCCA problem in $O(I(k,n)+n\log n)$ time, where $I(k,n)$ is the time complexity for the k-BCI problem.

We remark here that the k-BCI problem was also independently considered by Aggarwal et al.[2], in which it is called K maximum cliques in an interval graph. They make use of the matrix searching technique[1] and obtain an $O(min(k, \sqrt{k \log n})n \log\log n)$ algorithm, assuming the intervals are already sorted. For $k = o(\log n/(\log\log n)^2)$, our algorithm is asymptotically faster. It is not clear that their approach, which is entirely graph-theoretical, can be adapted to handle the k-BCCA problem directly. Since our solution to k-BCCA problem depends on our solution to k-BCI problem we present the k-BCI algorithm below. Our algorithm is based on the dynamic programming method.

2 k Best Cuts for Intervals

We first assume that there exists an angular position, say 0, on the circle at which no demand arc contains the point. We now "open" the circle at angle 0 and treat it as an angular interval $B = [0, 2\pi)$ whose CW end is 0 and CCW end is 2π. We still use terms CW end and CCW end to refer to the endpoints of the given angular intervals in the following discussion. In our convention, "CW" means "left" as applied to intervals and that "CCW" means "right". Denote by $C = \{c_1, c_2, \ldots, c_t\}$ the set of cut intervals. We that $c_1, c_2, \ldots, c_j$ form an increasing (angular order) sequence. We shall refer to the total weight of arcs covered by a set of cut intervals as *the total weight covered* by the cut intervals.

Note that when $k = 1$ the k-BCI problem is the conventional *maximum weight clique* problem and can be solved in $\Theta(n \log n)$ time[9]. When $k > 1$, it is not difficult to see that the *greedy* method, i.e., repeat for k times to find a maximum weight clique of the set of intervals and remove them, does not necessarily produce an optimal solution. We shall adopt a *dynamic programming* technique to tackle this problem.

For both of the k-BCI and the k-BCCA problems we define the following. A *cut set* $C_j^i \subseteq \{c_1, c_2, \ldots, c_j\}$ is a set of i cut intervals, where $1 \le i \le j$, such that: (1) $c_j \in C_j^i$, and (2) the selection of the other $i-1$ cut intervals in C_j^i from $\{c_1, c_2, \ldots, c_{j-1}\}$ is optimal in that C_j^i covers the most weight. In other words, C_j^i is the optimal cut set consisting of c_j and $i-1$ cuts in $\{c_1, c_2, \ldots, c_{j-1}\}$. Note that $C_j^1 = \{c_j\}$. It is clear that one of $C_j^k, j = k, \ldots, t$ must be an answer to the k-BCI problem. Denote $\#C_j^i$ as the total weight covered by C_j^i, and *CW-cut*(C_j^i) as the CW-most cut interval in C_j^i (c_j is the CCW-most cut interval in C_j^i). Define the *Weight Difference*, WD(C_r^l, c_j), of C_r^l with respect to c_j, where $1 \le l \le r < j$, as the total weight of arcs covered by C_r^l but not covered by c_j. We say that cut interval c_r is the *predecessor* of C_j^i, if WD(C_r^{i-1}, c_j) is the maximum among all possible $r \in [i-1, j-1]$. Consequently, C_j^i is the union of $\{c_j\}$ and C_r^{i-1}, and $\#C_j^i = \#C_j^1$ +WD(C_r^{i-1}, c_j), where c_r is the predecessor of C_j^i for $r \in [i-1, j-1]$. By definition WD(C_j^i, c_j) = WD(C_r^{i-1}, c_j), where c_r is the predecessor of C_j^i.

The basic idea of our k-BCI algorithm is: given $C_{i-1}^{i-1}, C_i^{i-1}, \ldots, C_t^{i-1}$, we can compute C_i^i, C_{i+1}^i $, \ldots, C_t^i$, with WD(C_r^{i-1}, c_j), for all $r \in [i-1, j-1]$, $j \in [i, t]$, computed on the fly. Suppose we have WD(C_r^{i-1}, c_{j-1}), for all $r \in [i-1, j-2]$. We show how to compute WD(C_r^{i-1}, c_j), for all $r \in [i-1, j-1]$, so that C_j^i can be computed. We do this in two steps:

(1) Compute WD(C_r^{i-1}, c_j) from WD(C_r^{i-1}, c_{j-1}) for all $r \in [i-1, j-1]$.

(2) Compute $\mathrm{WD}(C_j^{i-1}, c_j)$.

Call each arc whose CCW end lies between c_{j-1} and c_j a *departing arc at* c_j. If all θ_{ie}'s are distinct, then the arc whose CCW end determines c_{j-1} is the only departing arc for each j. Note that the difference between $\mathrm{WD}(C_r^{i-1}, c_j)$ and $\mathrm{WD}(C_r^{i-1}, c_{j-1})$ is the total weight of departing arcs covered by c_r. For step 1, we inspect every departing arc A_l and increment $\mathrm{WD}(C_r^{i-1}, c_{j-1})$ by w_l for every r such that c_r covers A_l, and after all departing arcs are considered, assign the resulting $\mathrm{WD}(C_r^{i-1}, c_{j-1})$ to $\mathrm{WD}(C_r^{i-1}, c_j)$, for all $r \in [i-1, j-1]$. For step 2, $\mathrm{WD}(C_j^{i-1}, c_j) = \mathrm{WD}(C_r^{i-2}, c_j)$, where c_r is the predecessor of C_j^{i-1}.

A trivial algorithm can, for every departing arc, spend $O(j)$ time to do the updating for step 1 and spend $O(1)$ time for step 2; and then spend $O(j)$ time to find the predecessor of C_j^i. Consequently, $C_j^i, j \in [i, t]$, can be computed in $O(n^2)$ time, and the k-BCI problem can be solved in $O(kn^2)$ time.

We shall explore in the following the *staircase* property of the values of Weight Differences for a fixed i, so that the updating for every departing arc can be done more efficiently.

Lemma 1 *If $WD(C_{r_1}^{i-1}, c_{j_1}) \leq WD(C_{r_2}^{i-1}, c_{j_1})$, where $r_1 < r_2 \leq j_1$, then for all $j_2 > j_1$, $WD(C_{r_1}^{i-1}, c_{j_2}) \leq WD(C_{r_2}^{i-1}, c_{j_2})$.*

Proof: It suffices to consider those arcs whose CCW ends lie between c_{j_1} and c_{j_2}. Since $r_1 < r_2 \leq j_1 < j_2$, for every such arc A_l, it is impossible that A_l intersects c_{r_1} but not c_{r_2}. Therefore, we only consider the following three possibilities: (1) A_l does not intersect either c_{r_1} or c_{r_2}, (2) A_l intersects c_{r_2} but not c_{r_1}, and (3) A_l intersects both. For case 1, the Weight Difference of $C_{r_1}^{i-1}$ and $C_{r_2}^{i-1}$ do not change. For case 2, the Weight Difference of $C_{r_2}^{i-1}$ is increased by w_l and for case 3, the Weight Differences of both $C_{r_1}^{i-1}$ and $C_{r_2}^{i-1}$ are increased by w_l. Thus $\mathrm{WD}(C_{r_2}^{i-1}, c_{j_2}) - \mathrm{WD}(C_{r_1}^{i-1}, c_{j_2}) \geq \mathrm{WD}(C_{r_2}^{i-1}, c_{j_1}) - \mathrm{WD}(C_{r_1}^{i-1}, c_{j_1})$ after all A_l's are considered. $\square$

If $\mathrm{WD}(C_{r_1}^{i-1}, c_{j_1}) \leq \mathrm{WD}(C_{r_2}^{i-1}, c_{j_1})$, and $r_1 < r_2 \leq j_1$, then $C_{r_2}^{i-1}$ is said to *dominate* $C_{r_1}^{i-1}$. Lemma 1 can be interpreted as follows: if $C_{r_2}^{i-1}$ dominates $C_{r_1}^{i-1}$, then $C_{r_1}^{i-1}$ can be eliminated from consideration because the Weight Difference of $C_{r_1}^{i-1}$ with respect to subsequent cut intervals can never exceed the Weight Difference of $C_{r_2}^{i-1}$. Hence when we are forming C_j^i, we only need to keep the collection of cut sets $C_{r_1}^{i-1}, C_{r_2}^{i-1}, \ldots, C_{r_v}^{i-1}$, where $r_1 < r_2 < \cdots < r_v$, such that: (1) $\mathrm{WD}(C_{r_1}^{i-1}, c_j) > \mathrm{WD}(C_{r_2}^{i-1}, c_j) > \cdots > \mathrm{WD}(C_{r_v}^{i-1}, c_j)$, and (2) for all $p \in [1, v]$, $\mathrm{WD}(C_{r_p}^{i-1}, c_j) \geq \mathrm{WD}(C_q^{i-1}, c_j)$ for all $r_{p-1} < q < r_p$ (assuming that $r_0 = 0$). Pictorially, if we look at the curve of $\mathrm{WD}(C_r^{i-1}, c_j)$ versus r for $1 \leq r \leq j-1$, they form a down-hill *staircase* (Figure 3); thus we call this the *staircase property* of the cut sets, and call each C_r^{i-1} on the staircase a *staircase cut set* associated with c_j. For a specific cut c_j, the relative position of two staircase cut sets associated with c_j is defined according to the positions of the CCW-most cut intervals in the two staircase cut sets. Thus for a $p \in [1, v]$, we call $C_{r_{p+1}}^{i-1}$ the CCW-neighbor of $C_{r_p}^{i-1}$, and $C_{r_{p-1}}^{i-1}$ the CW-neighbor of $C_{r_p}^{i-1}$. Among all staircase cut sets associated with c_j, the CW-most staircase cut set, i.e., $C_{r_1}^{i-1}$, has the largest $\mathrm{WD}(C_{r_1}^{i-1}, c_j)$, and c_{r_1} is the predecessor of C_j^i. Furthermore, an implication of Lemma 1 is that the CW-most staircase cut set associated with c_j is either the same as, or on the CCW-side of, the CW-most staircase cut set associated with c_{j-1}. Note also that the CCW-most staircase cut set associated with c_j is C_{j-1}^{i-1}, i.e., $r_v = j-1$.

Since $\mathrm{WD}(C_{r_p}^{i-1}, c_j)$ decreases monotonically with respect to increasing p, we can build a linked list L to store the collection of staircase cut sets. Each entry in L corresponds to a staircase cut

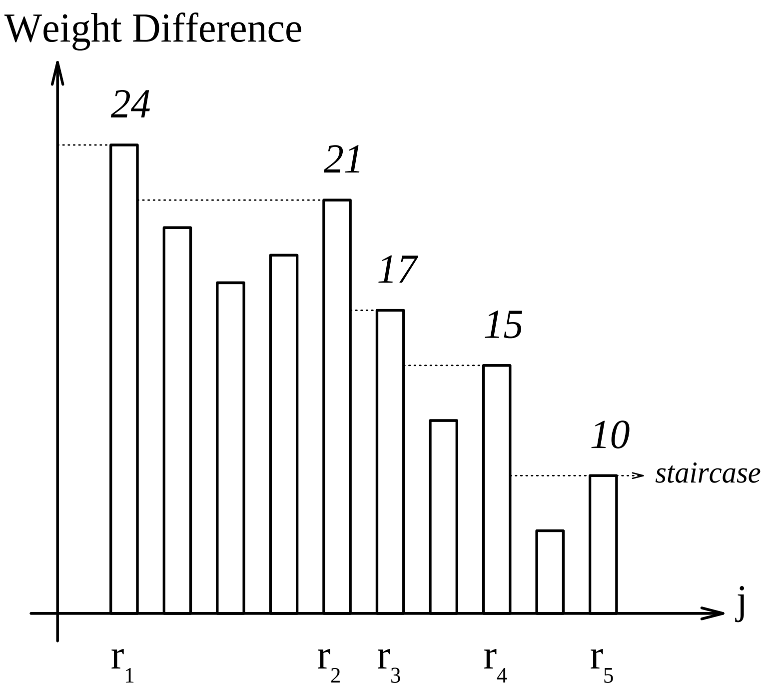

Figure 3: Down-hill staircase cut sets.

set $C^{i-1}_{r_p}$, and there are three fields stored in each entry in L: (1) the positional information of c_{r_p}, (2) a pointer to its CW-neighbor, $C^{i-1}_{r_{p-1}}$, and (3) $\Delta_{r_p} \equiv$ WD($C^{i-1}_{r_p}, c_j$)– WD($C^{i-1}_{r_{p+1}}, c_j$). For the CCW-most entry in L, $C^{i-1}_{r_v}$, let Δ_{r_v} =WD($C^{i-1}_{r_v}, c_j$). For the CW-most entry in L, $C^{i-1}_{r_1}$, assign its pointer to *nil*. Let MaxWD =WD($C^{i-1}_{r_1}, c_j$). One iteration of the algorithm refers to computing the best i-cuts based on the result of the best $(i-1)$-cuts, for $i = 2, \ldots, k$. More specifically, in iteration i, for each c_j, we compute the Weight Differences of the staircase cut sets with respect to c_j based on the Weight Differences of the staircase cut sets with respect to c_{j-1} so as to obtain C^i_j, where $j = i, \ldots, t$. In the beginning of iteration i, $j = i$, L contains only one entry, WD(C^i_i, c_i) =WD(C^{i-1}_{i-1}, c_i). Given that L represents $C^{i-1}_{r_1}, C^{i-1}_{r_2}, C^{i-1}_{r_3}, \ldots, C^{i-1}_{r_v}$, the staircase cut sets associated with c_{j-1}, we want to update L so that L represents the staircase cut sets associated with c_j. The following algorithm updates the staircase cut sets. We assume that $\#C^{i-1}_j$ and $\#C^1_j$ for every j have been computed in advance.

Algorithm 1 *Update-the-staircase-cut-set*

1. **For** every departing arc A_l, **do**
   (a) Increment Δ_{r_v} by w_ℓ.
   (b) Find the CCW-most staircase cut set, $C^{i-1}_{r_p}$, in L which does not cover A_l. If no such cut set exists, i.e., every staircase cut set in L covers A_l, then increment MaxWD by w_ℓ and skip Steps (c) and (d).
   (c) **While** $\Delta_{r_p} \leq w_\ell$ and $C^{i-1}_{r_p}$ is not the CW-most staircase cut set in L, **do**

       Increment $\Delta_{r_{p-1}}$ by Δ_{r_p}. Delete $C^{i-1}_{r_p}$ from L. Let $p = p - 1$.

(d) If $\Delta_{r_p} > w_\ell$ then decrement Δ_{r_p} by w_ℓ; otherwise decrement w_ℓ by Δ_{r_p}, delete $C^{i-1}_{r_p}$ from L, and increment MaxWD by w_ℓ.

2. Add a new entry C^{i-1}_j to L with $\mathrm{WD}(C^{i-1}_j, c_j) = \#C^{i-1}_j - \#C^1_j$; if necessary, delete any cut sets dominated by this new entry so that L represents the staircase cut sets. □

To speed up the computation of Algorithm 1, we shall employ the UNION-FIND algorithm[3] such that each deletion from L corresponds to a UNION operation, and each query operation in Step 1(b) corresponds to a FIND operation. Initially, every arc whose CW end is located between c_j and c_{j+1} belongs to a set S_j. Whenever a cut set $C^{i-1}_{r_p}$ is deleted from L, set S_{r_p} is merged into set $S_{r_{p-1}}$, i.e., $S_{r_{p-1}} = S_{r_{p-1}} \cup S_{r_p}$. During the entire operations of Algorithm 1, if some arc A_l belongs to set S_{r_p}, then $C^{i-1}_{r_p}$ does not cover A_l but $C^{i-1}_{r_{p+1}}$ does. Hence, it suffices to locate the staircase cut set described in Step 1(b) by a FIND operation.

In Step 2, we compute $\mathrm{WD}(C^{i-1}_j, c_j)$ according to the formula given earlier and insert it into L; one side effect of the insertion is that some CCW-side entries in L may be deleted if it is dominated by this new entry. It is better explained by an example. Suppose the four CCW-most entries in L are C^{i-1}_{j-10}, C^{i-1}_{j-5}, C^{i-1}_{j-3}, C^{i-1}_{j-1} (in CCW-order), and $\Delta_{j-10} = 5.0$, $\Delta_{j-5} = 2.0$, $\Delta_{j-3} = 3.0$, $\Delta_{j-1} = 1.0$. Then after inserting $\mathrm{WD}(C^{i-1}_j, c_j) = 7.1$, C^{i-1}_{j-1}, C^{i-1}_{j-3}, and C^{i-1}_{j-5} are to be deleted and Δ_{j-10} is changed to 3.9. These rules will guarantee that our invariant conditions are maintained, and the resulting location of cuts is correct. After L is updated to reflect the Weight Difference of staircase cut sets with respect to c_j, we simply use the combination of the CW-most entry in L and c_j to form C^i_j, and $\#C^i_j = \#C^1_j$+MaxWD. We summarize the above discussion in the following algorithm.

Algorithm 2 *k-BCI*

1. Open the circle at 0 and treat it as an angular interval $B = [0, 2\pi)$. Compute the set of cut intervals $C = \{c_1, c_2, \ldots, c_t\}$. Compute the total weight covered by every cut interval c_j in C, assign it to $\#C^1_j$.

2. **For** $i = 1, \ldots, k-1$, **do**

   (a) Initialize L to contain one entry, $\mathrm{WD}(C^i_i, c_i) = \#C^i_i - \#C^1_i$, and $j = i$.

   (b) **While** $(j < t)$ **do**

   Increment j by 1 and apply Algorithm 1 to update L such that it reflects the Weight Differences with respect to c_j; let $C^{i+1}_j = C^i_{r_1} \cup \{c_j\}$, and $\#C^{i+1}_j = \#C^1_j$+MaxWD, where $C^i_{r_1}$ is the CW-most entry in L.

3. Choose a $C^k_j, j \in [k, t]$ which covers the most weight as the solution to the problem. □

The correctness of the algorithm follows from the invariant conditions maintained through each iteration. The major time complexity is in Step 2. There are k iterations. For each iteration, the number of UNION operations is bounded by $t \leq n$, and the number of FIND operations is bounded by n. In general, the time spent on UNION-FIND operations is $O(n\alpha(n,n))$, where $\alpha(n,n)$ is the functional inverse of the Ackermann's function. However, the UNION operations in Algorithm 1 only take place for two adjacent entries; Gabow and Tarjan [6] showed that it takes linear time for this special case. Since every arc is deleted at most once, Step 2 can be attributed to $O(1)$ time per departing arc. Therefore, it takes $O(n)$ time in one iteration, giving $O(kn)$ time. If the endpoints are not sorted, the total time complexity of the k-BCI problem is $O(kn + n\log n)$. The space complexity depends on the output. If only the weight of covered arcs is of interest, then $O(n)$ is sufficient. If the actual locations of cuts need to be reported, then we have to use $O(kn)$ space to store the cut locations during the algorithm. However, the space complexity can be improved to $O(n)$, which is discussed in Section 4. We conclude:

Theorem 1 *The k-BCI problem can be solved in $O(kn + n\log n)$ time.*

Figure 4 shows the computing process with $k = 3$ for the example shown in Figure 2 assuming unit weight for each arc. The numbers enclosed in the rectangles are $\#C^i_j$, the numbers above the rectangles are Weight Differences of staircase cut sets and the circled numbers are Weight Differences of the CW-most staircase cut sets. The 3-BCI for this example is to place one cut in each of c_6, c_5, and c_3, such that 6 arcs are covered.

3 Restricted k Best Cuts for Intervals

To solve the k-BCCA problem, we introduce the k *Restricted Best Cuts for Intervals* (in short, k-RBCI) problem, which is a variant of k-BCI problem, but is more efficient and suitable than the k-BCI problem, in solving the k-BCCA problem. The definition of k-RBCI problem is given as follows. In addition to the set of intervals given in the original k-BCI problem, a set of *restricted ordered intervals*, $\mathcal{R} = \{s_1, s_2, \ldots, s_k\}$, is given, and it is required that the first cut in the solution (according to CCW-order) falls into s_1, the second cut falls into s_2, etc. A cut set D, called *restricted k-cuts*, is said to be *feasible* if the above requirement is met. Denote by $C = \{c_1, c_2, \ldots, c_t\}$ the set of cut intervals in CCW-order. Each restricted interval s_i can be represented by $[B_i, E_i]$, where B_i and E_i are indices of cut intervals in C.

Since $\mathcal{R}$ also implies an order of the solution, some restricted intervals can be confined further without affecting the solution. Let $B_0 = 0$ and $E_{k+1} = t + 1$. Then the confined restricted intervals are $[B'_i, E'_i]$, where

$$B'_i = \max_{0\le j\le i}(B_j - j) + i$$

$$E'_i = \min_{i\le j\le k+1}(E_j - j) + i$$

For instance, the k-BCI problem can be considered as a special case of k-RBCI problem by letting $\mathcal{R} = \{[1, t-k+1], [2, t-k+2], \ldots, [k, t]\}$. If there exists $B_i > E_i$ for some i, then there is no feasible solution. Otherwise, there exists an optimal solution. We assume that the k-RBCI problem is feasible and $[B_i, E_i]$ are the resulting confined restricted intervals.

Let F be an interval family and $\mathcal{R} = \{s_1, s_2, \ldots, s_k\}$ a set of restricted intervals. The following lemma shows that some intervals in F can be removed in finding an optimal k-RBCI-solution.

Lemma 2 *If $A_u \in F$ contains a restricted interval, or does not intersect with any restricted intervals, then D is the best restricted k-cuts for F iff D is the best restricted k-cuts for $F \setminus \{A_u\}$. In other words the removal of A_u from F does not affect the solution.*

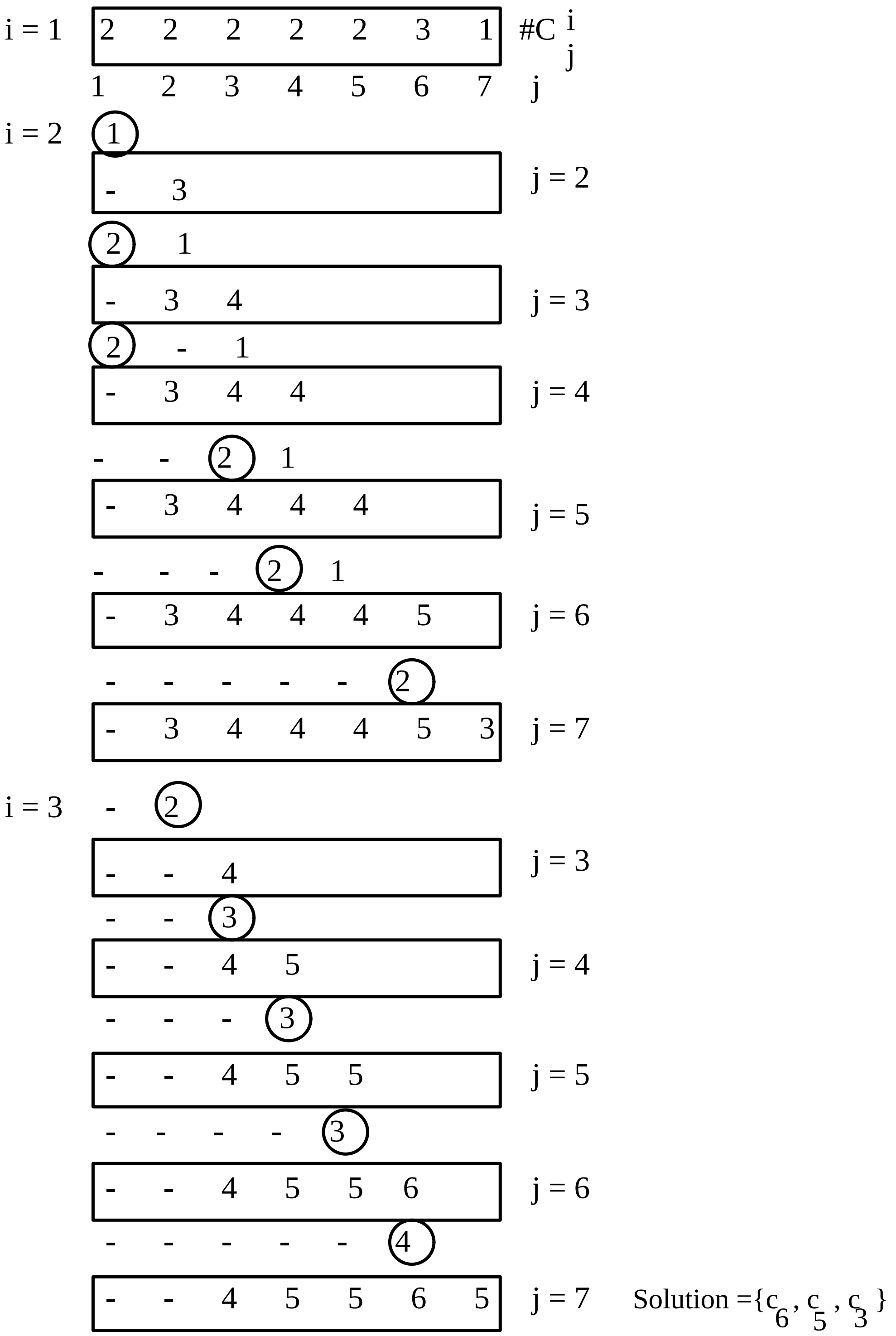

Figure 4: Computation process of 3-BCI for the example shown in Figure 2.

Proof: If A_u is an interval in F which contains some restricted interval, then A_u is covered by any feasible cut set. On the other hand, if A_u is an interval in H which does not intersect with any restricted intervals, then A_u is not covered by any feasible cut set. Therefore, the removal of A_u from F does not affect the solution. □

The corollary below follows from Lemma 2.

Corollary 1 *Let $H \subseteq F$ be the set of intervals such that an endpoint of each interval is contained in $s_i \in \mathcal{R}$ for some i. A cut set D is the best restricted k-cuts for F iff D is the best restricted k-cuts for H.*

The domination relationship as well as the staircase property of the k-BCI problem also hold here. Below, we describe some necessary change of Algorithm k-BCI so that it can be applied to k-RBCI problem. (1)The initialization of L (Step 2(a) in Algorithm 2) should be changed. (2)The computation of the predecessor of C_j^i needs modifications since the range of possible predecessors is different. The possible predecessor range of C_j^i, $j \in [B_i, E_i]$, is $[B_{i-1}, \min(E_{i-1}, j-1)]$. That is, only if $B_{i-1} \le r \le \min(E_{i-1}, j-1)$, is c_r a possible predecessor of C_j^i. We now show changes of the initialization of L, assuming that all irrelevant arcs, as defined in Lemma 2 have been removed.

For simplicity, let $b = B_{i-1}$ and $e = \min(E_{i-1}, B_i - 1)$. We describe the calculations of $WD(C_r^{i-1}, c_j)$ for every $r \in [b, e]$. Note that $j \in [B_i, E_i]$ and that initially $j = B_i$. Call each arc whose CW (CCW, resp.) end lies between c_b and c_e an *entering arc* (a *departing arc*, resp.) in $[c_b, c_e]$. Note that $\#C_v^{i-1}$'s, $v \in [B_{i-1}, E_{i-1}]$, are already computed. To calculate the weight difference, we need to subtract from $\#C_r^{i-1}$ the weight of arcs which are also covered by c_j. Let A_u be an arc covered by c_j. If A_u is an entering arc between c_e and c_j, it has no contributions to any $WD(C_r^{i-1}, c_j)$. And if A_u is not an entering arc between c_b and c_j, then A_u is covered by c_j and every feasible $(i-1)$it st cut. We need to decrement every $\#C_r^{i-1}$ the weight of A_u. However, if we only care about the location of the predecessor cut of C_j^i, this type of arc can also be eliminated from consideration. Thus we only need to consider entering arcs in $[c_b, c_e]$ which are also covered by c_j. Let $WD(C_r^{i-1}, c_j)$ represent $\#C_r^{i-1}$ minus the total weight of entering arcs in $[c_b, c_r]$ which are also covered by c_j. The time complexity of initialization takes $O(e - b + T)$ time, where T is the number of entering arcs between c_b and c_e whose weight is covered by c_{B_i}.

Lemma 3 *It takes $O(n + D)$ time to solve the k-RBCI problem, where D is the total number of departing arcs in the restricted intervals.*

Proof: The preprocessing phase of removing irrelevant arcs (Lemma 2) can be done in $O(n)$ time. After removal of these arcs, we have that the number of entering arcs E and the number of departing arcs D in the restricted intervals are of the same order, i.e., $E = O(D)$. The possible *initial* range of the predecessor cut of C_j^i, $j \in [B_i, E_i]$, is $[B_{i-1}, \min(E_{i-1}, B_i - 1)]$. These ranges are all disjoint for all i. Therefore the total time for initialization is $O(n)$. The factor D comes from the fact that each departing arc is charged one unit of time in maintaining the linked list. Thus the total complexity is as claimed. □

Corollary 2 *If the restricted intervals of the k-RBCI problem are pairwise disjoint, then k-RBCI problem can be solved in $O(n)$ time.*

Proof: In this case, D, the total number of departing arcs in all restricted intervals is $O(n)$. □

4 Interleaving and Monotonicity Property

In Section 2, we mentioned that it needs $O(n)$ space to report the actual positions of the best k-cuts for an interval family. However, we may apply k-RBCI to reduce its space complexity to $O(n)$ while the time complexity remains $O(kn + n \log n)$. To solve the k-BCI problem, we first solve the 1-RBCI problem, and then 2-RBCI problem, $\cdots$, k-RBCI problem. We shall show that the solution of the last k-RBCI problem is also a solution of the original k-BCI problem. This is based on a key lemma (Lemma 6 *interleaving property*). However, before we describe the property, we show some observations on the set of best k-cuts for an interval family.

Let the angular position of c_i (c_j, c_r, resp.) be θ_i (θ_j, θ_r, resp.). Then, $\theta_i < \theta_j$ and $c_i < c_j$ are used interchangeably. Let H be a family of intervals, and b and e be two points. Denote by $H_{b,e}$ the subfamily of H composed of those intervals contained in (b, e). If no confusion can arise we will say that a cut set is a best k-cuts on (b, e) if it consists of the k best cuts for $H_{b,e}$. In this section, we assume that b_i, e_i, b_j, and e_j are four points, and $\mathcal{I} = \{c_{i_1}, c_{i_2}, c_{i_3}, \ldots, c_{i_k}\}$ is a best k-cuts on (b_i, e_i).

Lemma 4 *If $b_i = b_j$, $e_i < e_j$ and there exists a best k-cuts on (b_j, e_j), $\mathcal{J} = \{c_{j_1}, c_{j_2}, c_{j_3}, \ldots, c_{j_k}\}$, such that $j_k < i_k$, then $\mathcal{I}$ is also a best k-cuts on (b_j, e_j).*

Proof: Let $H_1 = H_{b_j,e_j} \setminus H_{b_i,e_i}$. We partition H_1 into two subfamilies H_2 and H_3 such that H_2 is composed of those intervals covered by $\mathcal{J}$. Let $H' = H_{b_j,e_j} \setminus H_3$. Since no interval in H_3 is covered by $\mathcal{J}$, any best k-cuts for H' is also a best k-cuts for H_{b_j,e_j}. On the other hand, $H' = H_{b_i,e_i} \cup H_2$. As $j_k < i_k$, every interval in H_2 is covered by i_k. Hence, $\mathcal{I}$ is also a best k-cuts for H' as well as a best k-cuts on (b_j, e_j). $\Box$

Lemma 5 *If $b_i = b_j$ and $e_i < e_j$, then there exists a best k-cuts on (b_j, e_j), $\{c_{j_1}, c_{j_2}, c_{j_3}, \ldots, c_{j_k}\}$, such that $c_{i_r} \leq c_{j_r}$, for any $r \in [1, k]$.*

Proof: If $j_k < i_k$, by Lemma 4, $\mathcal{I}$ is also a best k-cuts on (b_j, e_j), and we are done. Therefore, assume $j_k \geq i_k$. Since $\{c_{j_1}, c_{j_2}, c_{j_3}, \ldots, c_{j_{k-1}}\}$ is a best $(k-1)$-cuts on (b_j, c_{j_k}), one can show that $c_{i_r} \leq c_{j_r}$, for any $r \in [1, k-1]$ by induction. $\Box$

Lemma 6 (Interleaving Property) *Let $\{c_{i_1}, c_{i_2}, c_{i_3}, \ldots, c_{i_k}\}$ be a best k-cuts for H. There exists a best $(k+1)$-cuts for H, $\{c_{j_1}, c_{j_2}, c_{j_3}, \ldots, c_{j_{k+1}}\}$, such that $c_{j_r} \leq c_{i_r} \leq c_{j_{r+1}}$, for any $r \in [1, k]$.*

Proof: $\{c_{i_1}, c_{i_2}, c_{i_3}, \ldots, c_{i_k}\}$ is a best k-cuts on $(-\infty, \infty)$. Since $\{c_{j_1}, c_{j_2}, c_{j_3}, \ldots, c_{j_k}\}$ is a best k-cuts on $(-\infty, c_{j_{k+1}})$, and $c_{j_{k+1}} < \infty$, we may assume that $c_{j_r} \leq c_{i_r}$, for any $r \in [1, k]$ (by Lemma 5). If $c_{i_r} \leq c_{j_{r+1}}$, for any $r \in [1, k]$, then we are done. Otherwise, let r be the smallest integer such that $c_{i_r} > c_{j_{r+1}}$. We know that $\{c_{i_r}, c_{i_{r+1}}, c_{i_{r+2}}, \ldots, c_{i_k}\}$ is a best $(k-r+1)$-cuts on $(c_{i_{r-1}}, \infty)$, and $\{c_{j_{r+1}}, c_{j_{r+2}}, c_{j_{r+3}}, \ldots, c_{j_{k+1}}\}$ is a best $(k-r+1)$-cuts on (c_{j_r}, ∞). Since $c_{i_r} > c_{j_{r+1}}$, by the same arguments in Lemma 4, one can show that $\{c_{i_r}, c_{i_{r+1}}, c_{i_{r+2}}, \ldots, c_{i_k}\}$ is a best $(k-r+1)$-cuts on (c_{j_r}, ∞). Hence, $\{c_{j_1}, c_{j_2}, c_{j_3}, \ldots, c_{j_r}, c_{i_r}, c_{i_{r+1}}, c_{i_{r+2}}, \ldots, c_{i_k}\}$ is a best $(k+1)$-cuts for H. $\Box$

Theorem 2 *The k-BCI problem can be solved in $O(kn + n \log n)$ time and $O(n)$ space.*

Proof: We first solve the 1-BCI problem. Based on the output, we then solve the 2-RBCI problem, whose solution is also a best 2-cuts. We keep this iteration until the best k-cuts is found. In iteration w, we assume that the best $(w-1)$-cuts, $\mathcal{I} = \{c_{i_1}, c_{i_2}, c_{i_3}, \ldots, c_{i_{w-1}}\}$, is found. By Lemma 6, we can solve the w-RBCI problem with respect to the set of restricted intervals $\mathcal{R} = \{(-\infty, c_{i_1}], [c_{i_1}, c_{i_2}], \ldots, [c_{i_{w-1}}, \infty)\}$. As the restricted intervals only share endpoints, every cut not in $\mathcal{I}$ only belongs to one restricted interval, and every cut in $\mathcal{I}$ belongs to 2 restricted intervals. Therefore, it takes $O(1)$ space for each cut to remember its predecessor. Hence, it takes $O(n)$ time and $O(n)$ space in one iteration, giving $O(kn)$ time in total. If endpoints are not sorted, we sort them in $O(n \log n)$ time. $\Box$

Lemma 5 can be generalized to the following lemma, which is an important property and will be used in the next section.

Lemma 7 (Monotonicity Property) *If $b_i \leq b_j$ and $e_i \leq e_j$, then there exists a best k-cuts on (b_j, e_j), $\{c_{j_1}, c_{j_2}, c_{j_3}, \ldots, c_{j_k}\}$, such that $c_{i_r} \leq c_{j_r}$, for any $r \in [1, k]$.*

Proof: By Lemma 5, there exists a best k-cuts on (b_i, e_j), $\{c_{p_1}, c_{p_2}, c_{p_3}, \ldots, c_{p_k}\}$, such that $c_{i_r} \leq c_{p_r}$, for any $r \in [1, k]$. By the same arguments in Lemma 5, there exists a best k-cuts on (b_j, e_j), $\{c_{j_1}, c_{j_2}, c_{j_3}, \ldots, c_{j_k}\}$, such that $c_{p_r} \leq c_{j_r}$, for any $r \in [1, k]$. $\Box$

5 k Best Cuts for Circular Arcs

Now let us turn back to the k-BCCA problem. The objective is to find the best k-cuts for F, a family of circular-arcs. The algorithm *k-BCI* will be used as a subroutine. Note that the solution of k-BCI is not necessarily unique. The algorithm we proposed only returns *one* optimal solution. The possibility of existence of many optimal solutions comes from the fact that a cut set C_j^i may have many predecessors. To help the design of k-BCCA algorithm, we need a more specific optimal k-BCI solution. We call it the *"CW-most"* optimal solution. That is, in the selection of predecessor cut, we always choose the CW-most one if there is more than one. In other words, the staircase cut set is maintained in a non-increasing order (instead of strictly decreasing order.) We assume that CW-most solution will be returned whenever k-BCI or k-RBCI algorithm is invoked. Note that for any two CW-most solutions, the monotonicity property still holds.

Suppose c_i is a cut interval with angle θ_i. Call it a *"starting cut"*. Let us remove the arcs covered by c_i, open the unit circle at θ_i, and treat it as an angular interval $[\theta_i, \theta_i + 2\pi)$. Then, the arcs not covered by c_i form a family of intervals contained in $[\theta_i, \theta_i + 2\pi)$, denoted by F_i. By applying the algorithm *$(k-1)$-BCI*, we obtain the best $(k-1)$-cuts, denoted by $D_i = \{c_{i_1}, c_{i_2}, \ldots, c_{i_{k-1}}\}$ for F_i.

The starting cut c_i is said to be *"good"* if c_i is contained in some optimal k-cuts for F. For simplicity, let c_{i_k} denote the cut interval at angular position $\theta_i + 2\pi$. In circular-arcs model, c_{i_k} and c_i are identical. However, they are considered different in intervals model, since they are at different angular positions.

Fact 2: If c_i is good, then $D_i \cup \{c_i\}$ is the best k-cuts for F.

Note that c_i is defined in F, a circular-arc family, and the angular position of c_i is contained in $[0, 2\pi]$. However, since F_i is a family of intervals contained in $[\theta_i, \theta_i + 2\pi)$, the angular position of each cut in D_i is also contained in $[\theta_i, \theta_i + 2\pi)$, and therefore contained in $[0, 4\pi)$. However, the best k-cuts for F can be obtained by taking modulo 2π. In this section, $\theta_i < \theta_j$ and $c_i < c_j$ are used interchangeably, and we say c_r is in $[c_i, c_j]$ if $\theta_i \leq \theta_r \leq \theta_j$.

Lemma 8 *Let c_i and c_j be two cut intervals (with angles θ_i and θ_j), $D_i = \{c_{i_1}, c_{i_2}, \ldots, c_{i_{k-1}}\}$ and $D_j = \{c_{j_1}, c_{j_2}, \ldots, c_{j_{k-1}}\}$ be the corresponding $(k-1)$-BCI solutions. Denote by θ_{i_r} (θ_{j_r}, resp.) the angle of the cut c_{i_r} (c_{j_r}, resp.). If $\theta_i < \theta_j$, then $\theta_{i_r} \leq \theta_{j_r}$, for any $r \in [1, k-1]$.*

Proof: If $\theta_i < \theta_j$, then $\theta_i + 2\pi < \theta_j + 2\pi$. By Lemma 7, $\theta_{i_r} \leq \theta_{j_r}$, for any $r \in [1, k-1]$. □

Fact 2 implies that once a good cut interval is located, the k-BCCA problem can be reduced to the k-BCI problem. A trivial algorithm can apply t-times k-BCI algorithm to try every cut interval as a starting cut. Instead, we pick arbitrarily a cut c_i as the starting cut, and apply k-BCI algorithm to obtain the corresponding $(k-1)$-BCI solution, $D_i = \{c_{i_1}, c_{i_2}, \ldots, c_{i_{k-1}}\}$. Lemmas 9 and 10 enable us to confine our search for a good cut. Note that although c_i is picked arbitrarily, it remains fixed throughout this section.

Lemma 9 *There exists a good cut in $[c_{i_r}, c_{i_{r+1}}]$, for each $0 \leq r < k$.*

Proof: Suppose c_i is not a good cut. Let c_j be the first good cut encountered in a CW-traversal starting from c_i. By Fact 2, $\{c_j, c_{j_1}, c_{j_2}, \ldots, c_{j_{k-1}}\}$ is the best k-cuts for F. Hence, c_{j_1} is a good cut. By Lemma 8, $\theta_{j_r} \leq \theta_{i_r}$, $0 < r < k$. By the definition of c_j, $\theta_{j_1} > \theta_i$. Hence, c_{j_1} is a good cut in $[c_i, c_{i_1}]$. Now, open the circle at $c_p = c_{j_1}$, and let $D_p = \{c_{p_1}, c_{p_2}, \ldots, c_{p_{k-1}}\}$ be the corresponding $(k-1)$-BCI solution. Both $\{c_j, c_{j_1}, c_{j_2}, \ldots, c_{j_{k-1}}\}$ and $D_p \cup \{c_p\}$ are best k-cuts for F. Since the algorithm k-BCI returns the CW-most solution, we have $c_{p_r} \leq c_{j_{r+1}}$, for $0 < r < k$. Therefore, $c_{p_r} \leq c_{j_{r+1}} \leq c_{i_{r+1}}$. On the other hand, since $c_i < c_p$, $c_{i_r} \leq c_{p_r}$. Hence, a good cut c_{p_r} is contained in $[c_{i_r}, c_{i_{r+1}}]$. □

Lemma 9 implies that it suffices to search only in $[c_i, c_{i_1}]$ to locate a good cut. Below, we show that the search range can be confined further. Let c_j be a cut in $[c_i,\ c_{i_1}]$, and $D_j = \{c_{j_1}, c_{j_2}, \ldots, c_{j_{k-1}}\}$ be the corresponding $(k-1)$-BCI solution. Cut c_j in $[c_i, c_{i_1}]$ is said to be *"redundant"* if $\theta_{j_{k-1}} \geq \theta_i + 2\pi$ and *"non-redundant"* otherwise.

Lemma 10 *There exists a good cut in $[c_i, c_{i_1}]$ which is non-redundant.*

Proof: Suppose c_i is not a good cut. Let c_j be the first good cut encountered in a CCW-traversal starting from c_i. The cut $c_{j_{k-1}}$ is good as $c_{j_{k-1}} \in D_j$ and $D_j \cup \{c_j\}$ is the best k-cuts for F. We have $\theta_{j_{k-1}} < \theta_j + 2\pi$. If c_j is redundant, then $\theta_i + 2\pi < \theta_{j_{k-1}} < \theta_j + 2\pi$. Thus, in a CCW-traversal starting from c_i, the first good cut encountered would have been $c_{j_{k-1}}$ instead of c_j, a contradiction. □

Let c_j be a cut in $[c_i, c_{i_1}]$. By Lemma 8, if c_j is non-redundant, then every cut in $[c_i, c_j]$ is non-redundant. On the other hand, if c_j is redundant, then every cut in $[c_j, c_{i_1}]$ is redundant. Therefore, there exists a *"threshold"* cut c_h in $[c_i, c_{i_1}]$, such that every cut in $[c_i,\ c_h]$ is non-redundant, and every cut in $(c_h, c_{i_1}]$ is redundant. We can use binary search to locate the threshold cut c_h, and Lemma 10 guarantees that a good cut exists in $[c_i, c_h]$.

Lemma 11 *Let $c_j = c_{i_1}$ be a starting cut, and $D_j = \{c_{j_1}, c_{j_2}, \ldots, c_{j_{k-1}}\}$ be the corresponding $(k-1)$-BCI solution. If we define $c_{i_k} = c_i$ and $c_{i_{k+1}} = c_{i_1}$, then $c_{i_r} \leq c_{j_r} \leq c_{i_{r+2}}$, for $0 < r < k$.*

Proof: Since $c_i < c_j$ and $c_i + 2\pi < c_j + 2\pi$, by the monotonicity property (Lemma 7), we have $c_{i_r} \leq c_{j_r}$, for $0 < r < k$. To show the upper bound of c_{j_r}, we consider two cases:

Case 1. Suppose c_i is a good cut. Then, $c_j = c_{i_1}$ is also a good cut. Since we assume that the k-BCI algorithm returns the CW-most optimal solution. $c_{j_r} \leq c_{i_{r+1}} < c_{i_{r+2}}$, for $0 < r < k$.

Case 2. Suppose c_i is not a good cut. Let c_l be the first good cut encountered in a CCW-traversal starting from c_i, and $D_l = \{c_{l_1}, c_{l_2}, \ldots, c_{l_{k-1}}\}$ be the corresponding $(k-1)$-BCI solution. Then, c_l must be non-redundant, and $c_{l_r} \leq c_{i_{r+1}}$. Let $c_l = c_{l_k}$. Now, consider another $(k-1)$-BCI solution corresponding to a starting cut $c_{l_1} = c_m$, $D_m = \{c_{m_1}, c_{m_2}, \ldots, c_{m_{k-1}}\}$. Since both $D_l \cup \{c_l\}$ and $D_m \cup \{c_m\}$ are the best k-cuts for F and the k-BCI algorithm always returns CW-most optimal solution, $c_{m_r} \leq c_{l_{r+1}}$, for $0 < r < k$. Since $c_j = c_{i_1} \leq c_{l_1} = c_m$, $c_{j_r} \leq c_{m_r} \leq c_{l_{r+1}} \leq c_{i_{r+2}}$, for $0 < r < k$. □

Lemma 12 *The binary search for the threshold cut takes $O(n \log n)$ time.*

Proof: To test whether a cut c_j in $[c_i, c_{i_1}]$ is redundant or not, one can solve the $(k-1)$-RBCI problem with respect to the set of restricted intervals $\mathcal{R} = \{[c_{i_1}, c_{i_3}], [c_{i_2}, c_{i_4}], \ldots, [c_{i_{k-2}}, c_{i_k} = c_i + 2\pi], [c_{i_{k-1}}, c_{i_1} + 2\pi]\}$. Since both the summation of the number of departing arcs contained in each restricted interval and the number of intervals are $O(n)$, the test takes $O(n)$ time. The binary search may invoke such a test at most $O(\log n)$ times. Hence, it takes $O(n \log n)$ time. □

For any cut c_j in $[c_i, c_{i_1}]$, Lemma 8 states that θ_{i_r} is a lower bound for θ_{j_r}. The next lemma shows that there also exists an upper bound for θ_{j_r}, if c_j is non-redundant.

Lemma 13 *Let c_j be a non-redundant cut in $[c_i, c_{i_1}]$. Then cut c_{j_r} is contained in $[c_{i_r}, c_{i_{r+1}}]$, for any $r \in [1, k-1]$.*

Proof: By definition, $\theta_{j_{k-1}} < \theta_i + 2\pi = \theta_{i_k}$. By Lemma 8, $\theta_{i_r} \leq \theta_{j_r}$, for any $r \in [1, k-1]$. It thus remains to prove $\theta_{j_r} \leq \theta_{i_{r+1}}$, for any $r \in [1, k-2]$. Let F' be the set of interval contained in $[\theta_{i_1}, \theta_{i_k}]$, and F'' be the set of intervals contained in $[\theta_j, \theta_{j_{k-1}}]$. The cut set $\{c_{i_2}, c_{i_3}, \ldots, c_{i_{k-1}}\}$ is the best $(k-2)$-cuts for F', and the cut set $\{c_{j_1}, c_{j_2}, \ldots, c_{j_{k-2}}\}$ is the best $(k-2)$-cuts for F''. Since $\theta_j \leq \theta_{i_1}$, and $\theta_{j_{k-1}} < \theta_{i_k}$, $\theta_{j_r} \leq \theta_{i_{r+1}}$, for any $r \in [1, k-2]$, by Lemma 8. □

Recall that Lemma 10 implies that there exists a good cut between c_i and c_h. Next, we show how to derive D_j for every c_j in $[c_i, c_h]$. Our computation can be divided into several phases. In Phase 1, we derive D_j, where c_j is the median cut between c_i and c_h. In Phase 2, we derive two cut sets, D_p and D_q, where c_p and c_q are respectively the median cuts between c_i and c_j, and between c_j and c_h. In Phase 3, we derive 4 cut sets, corresponding to the median cuts between (c_i, c_p), (c_p, c_j), (c_j, c_q) and (c_q, c_h) respectively. In general, in Phase r, we need to derive 2^{r-1} cut sets, and each of them can be formulated as a k-RBCI problem. Since the total number of cuts in $[c_i, c_h]$ is less than n, the number of phases is bounded by $O(\log n)$. We show below that the time spent in each phase is bounded by $O(n)$.

Recall that in Phase 1, we need to derive cut set D_j, the $(k-1)$-BCI solution corresponding to c_j, where c_j is the median cut between $[c_i, c_h]$. Note that D_j is feasible and also the best restricted $(k-1)$-cuts for F_j with respect to the restricted intervals, $\mathcal{R}_j = \{[c_{i_1}, c_{h_1}], [c_{i_2}, c_{h_2}], \ldots, [c_{i_{k-1}}, c_{h_{k-1}}]\}$ since c_{j_r} is in $[c_{i_r}, c_{h_r}]$, for any $r \in [1, k-1]$ (Lemma 13.) Let $H_j \subseteq F_j$ be constructed as in Corollary 1. Then D_j is the best $(k-1)$-cuts for H_j.

Now, consider Phase 2. We need to derive D_p and D_q, the $(k-1)$-BCI solutions corresponding to c_p and c_q respectively. If we let $\mathcal{R}_p = \{[c_{i_1}, c_{j_1}], [c_{i_2}, c_{j_2}], \ldots, [c_{i_{k-1}}, c_{j_{k-1}}]\}$ be the set of restricted intervals, and $H_p \subseteq F_p$ be constructed by Corollary 1, then D_p is the best restricted $(k-1)$-cuts for H_p. To derive D_q, we let $\mathcal{R}_q = \{[c_{j_1}, c_{h_1}], [c_{j_2}, c_{h_2}], \ldots, [c_{j_{k-1}}, c_{h_{k-1}}]\}$ be the set of restricted intervals. Notice that the restricted intervals in F_p and F_q are disjoint except at the endpoints.

In general, suppose we shall derive the cut sets corresponding to $c_p^1, c_p^2, \ldots, c_p^w$ in some phase. As described before, we can solve the k-RBCI-problems for $H_p^1, H_p^2, \ldots, H_p^w$ instead. Notice that the restricted intervals in each RBCI-problem are disjoint except for the endpoints. By Corollary 1, an arc A_u in F appears in some H_p^r if and only if an endpoint of A_u is contained in some restricted interval $\mathcal{R}_p^r$. For each endpoint of A_u, A_u may appear in one of $H_p^1, H_p^2, \ldots, H_p^w$, or two if the endpoint coincides with an endpoint of some restricted interval. Therefore A_u may appear in at most four of $H_p^1, H_p^2, \ldots, H_p^w$. Therefore, the total number of intervals in $H_p^1, H_p^2, \ldots, H_p^w$ is bounded by $4n$. Now, consider the number of cut intervals contained in the restricted intervals. Since the restricted intervals in each k-RBCI-problem are disjoint except for the endpoints, the total number of entering arcs and cut intervals in those RBCI-problem is bounded by $O(n)$.

Lemma 14 *The computation time in each phase is bounded by $O(n)$.*

Proof: Since both the summation of the number of intervals and the summation of the number of cut intervals contained in the restricted interval of the RBCI-problems in the same phase are bounded by $O(n)$. The computation time in each phase is bounded by $O(n)$. $\square$

Note that the transformations to k-RBCI problems only determine the positions of the cut sets, and the maximum weight it covers on H, which is only a subset of F. It remains to determine how much weight it covers on the original set, F. In fact, we choose to calculate the weight not covered by $\{c_j\} \cup D_j$ first, and then subtract it from the total weight of circular-arcs to obtain the weight covered by $\{c_j\} \cup D_j$. Consider the cut set $\{c_j\} \cup D_j = \{c_j, c_{j_1}, c_{j_2}, \ldots, c_{j_{k-1}}\}$. For simplicity, let c_j, c_{j_0}, and c_{j_k} represent the same cut in circular-arcs model. Let w_{j_r} be the total weight of circular-arcs contained in $(\theta_{j_r}, \theta_{j_{r+1}})$, for any $r \in [0, k-1]$. Then, the total weight not covered by $\{c_j\} \cup D_j$ is $\Sigma_{r \in [0,k-1]} w_{j_r}$.

Suppose the number of non-redundant cuts is l. Without loss of generality, assume the non-redundant cuts are $c_1, c_2, \ldots, c_l$, in CCW-order. By Lemma 13, $\theta_{1_r} \leq \theta_{2_r} \leq \cdots \theta_{l_r} \leq \theta_{1_{r+1}}$, for any $r \in [0, k-1]$. We show the calculation of w_{j_r} for any $j \in [1, l]$ and $r \in [0, k-1]$ in Algorithm 3.

In Algorithm 3, variables $ccwp$ and cwp represent two points in the circle, and these two points are moving independently in counterclockwise direction. There are 6 different configurations for a circular-arc in relation to the locations of $ccwp$ and cwp (Figure 5). We maintain 6 lists, each of which corresponds to a different configuration. Initially, cwp is at θ_i, and $ccwp$ at θ_{i_1}. Whenever $ccwp$ (or cwp) passes an endpoint of some circular-arc, this circular-arc is moved from one list to another according to the change of configuration. For example, List 1 (Figure 5(a)) contains those arcs which are contained in interval $(cwp, ccwp)$, and for every arc $i = (cw(i), ccw(i))$ in List 5 (Figure 5 (e)) we have $(ccwp, cw(i), cwp, ccw(i))$ when they are traversed counterclockwise, i.e., the CCW-end of arc i lies in interval $(cwp, ccwp)$. Suppose that arc i was in List 1. As soon as cwp is moved past the CW-end of arc i, arc i will be moved from List 1 to List 5. Note that the time spent on maintaining these lists is bounded by the number of endpoints $ccwp$ and cwp have encountered. The following iterations compute w_{j_r}, for every $j \in [1, l]$ and $r \in [0, k-1]$.

Algorithm 3 *Covered-Weight-Computation*

1. Let $cwp = \theta_i$, and $ccwp = \theta_{i_1}$. Assign each circular-arc to one of six lists according to its configuration.

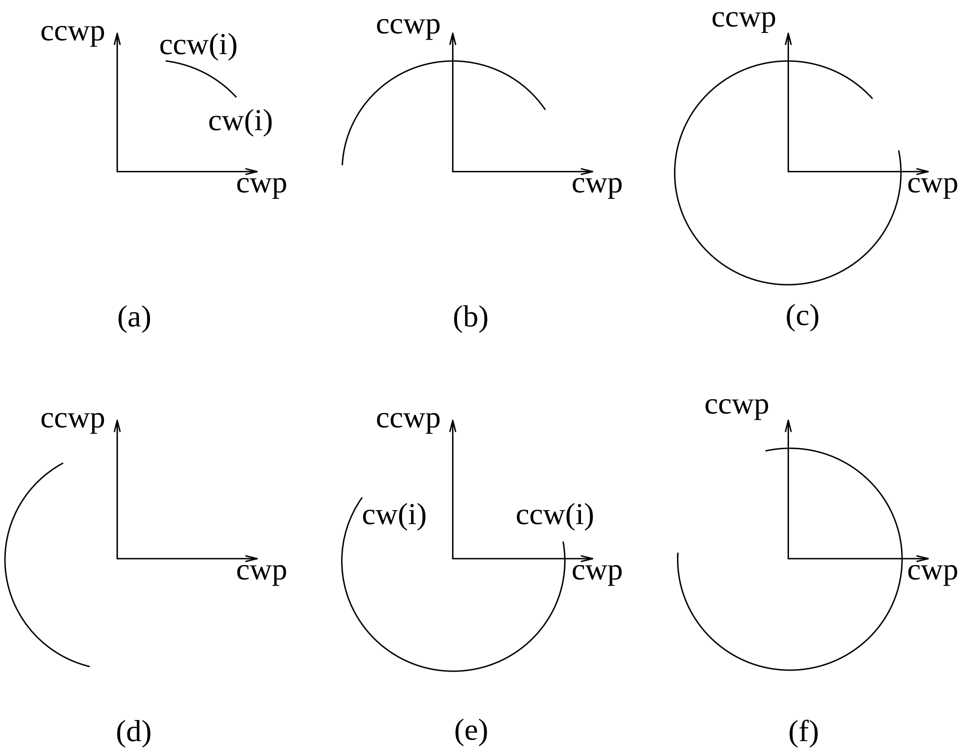

Figure 5: Six different configurations between two points and an arc.

2. **For** $r = 0$ to $k-1$ **do**

   For $j = 1$ to l **do**

   (a) Move (counterclockwise) cwp to θ_{j_r}, and update the six lists.

   (b) Move (counterclockwise) $ccwp$ to $\theta_{j_{r+1}}$, and update the six lists.

   (c) Let w_{j_r} be the total weight of circular-arc contained in $(cwp, ccwp)$.

□

Lemma 15 *Algorithm 3 takes $O(n+kl)$ time, where l is the number of non-redundant cuts.*

Proof: Both $ccwp$ and cwp traverse the circle only once, and the number of different w_{j_r} is $O(kl)$. □

The time complexity of Algorithm 3 can be reduced to $O(n)$ if $l = O(n/k)$. This can be achieved by a more careful selection of c_i. An approach is to apply k-BCI algorithm twice. The first time is just a probing. We start it by arbitrarily picking a cut interval c_p, and let $D_p = \{c_{p_1}, c_{p_2}, \ldots, c_{p_{k-1}}\}$ be the corresponding $(k-1)$-BCI solution. If we consider that the cuts in D_p divide the circle into k (closed) parts, then each part contains at least one good cut (Lemma 9.) Let the part which contains the least number of cut intervals be $[c_{p_r}, c_{p_{r+1}}]$. Since there are k parts, this part contains at most $t/k+1$ cut intervals. We then choose cut interval c_{p_r} as the starting cut (i.e., $c_i = c_{p_r}$,) and we apply k-BCI algorithm again to obtain $D_i = \{c_{i_1}, c_{i_2}, \ldots, c_{i_{k-1}}\}$. Note that c_{i_1} is not necessarily equal to $c_{p_{r+1}}$. However, only non-redundant cuts contained in $[c_{p_r}, c_{p_{r+1}}]$ need to be considered, and the number is bounded by $t/k+1$.

We now summarize our algorithm k-BCCA in the following.

Algorithm 4 *k-BCCA*

1. Arbitrarily select a cut c_p.
2. Find D_p, the best $(k-1)$-cuts for F_p.
3. Let $c_i = c_{p_r}$, where $[c_{p_r}, c_{p_{r+1}}]$ contains the least number of cut intervals for each $0 \le r < k$.
4. Find D_i, the best $(k-1)$-cuts for F_i.
5. Using binary search to locate the threshold cut c_h.
6. Let $\mathcal{C} = \{c_i, c_h\}$. **While** $\mathcal{C}$ does not contain every cut between $[c_i, c_h]$ **do**
   (a) Assume $\mathcal{C} = \{c_p^1, c_p^2, \ldots, c_p^w\}$ in CCW-order. Let $\mathcal{C}' = \{c_q^1, c_q^2, \ldots, c_q^{w-1}\}$ in CCW-order be the set of median cuts among $\mathcal{C}$. Let $\mathcal{R}_q^t = \{[c_{p1}^t, c_{p1}^{t+1}], [c_{p2}^t, c_{p2}^{t+1}], \ldots, [c_{p_{k-1}}^t, c_{p_{k-1}}^{t+1}]\}$, $t \in [1, w-1]$ be the set of restricted intervals. Let $H_q^t = \phi$, for $t \in [1, w-1]$.
   (b) For each circular-arc A_u, if an endpoint of A_u is contained in some interval in $\mathcal{R}_q^t$, then $H_q^t = H_q^t \cup \{A_u\}$.
   (c) For each $t \in [1, w-1]$, remove the circular-arcs covered by c_q^t from H_q^t, and open the circle at c_q^t. Calculate D_q^t, the best restricted $(k-1)$-cuts for H_q^t subject to the restricted intervals, $\mathcal{R}_q^t$.
   (d) Let $\mathcal{C} = \mathcal{C} \cup \mathcal{C}'$.
7. Apply Algorithm 3 to compute the total weight covered by $\{c_j\} \cup D_j$, for each c_j in $[c_i, c_h]$.
8. The set which covers the most weight is the solution. $\Box$

Theorem 3 The k-BCCA problem can be solved in $O(kn + n \log n)$ time.

Proof: Steps 1 through 4 take $O(kn)$ time. It takes $O(n \log n)$ time to locate the threshold cut in Step 5. Steps 6(a) through 6(d) constitute one phase. There are $O(\log n)$ phases, and each phase takes $O(n)$ time as proved in Lemma 14. Step 7 takes $O(n)$ time. Therefore, it takes $O(kn+n \log n)$ in total. $\Box$

6 Conclusion

We have solved the best k-cuts on circular-arc graphs. The circular-arcs which are covered by a cut form a clique in the circular-arc graph, we call it a linear clique. However, a clique in circular-arc graph may contain only three arcs which cover the circle and cannot be covered by a single cut. The k *Maximum Clique Cover (k-MQC)* problem is to find k cliques which cover the most arcs. In [8] [12] it is proved that to find the minimum number of cliques which cover a circular-arc family F, it suffices to use linear cliques only, provided that F itself is not a clique. Nevertheless, if F is a clique, the k-MQC problem is trivial. Suppose that F is not a clique. If $k = 1$, then the k-MQC problem is equivalent to the problem of finding the maximum clique on circular-arc graphs. Assume $k > 1$. Let F' be the family of circular-arcs which are covered by the optimal k cliques. Since F is not a clique and $k > 1$, F' can be covered by k linear cliques. Therefore, if $k > 1$, the k-MQC problem is equivalent to the k-BCCA problem, and thus can be solved in $O(kn + n \log n)$ time. Furthermore, since k-BCCA algorithm only apply k-BCI algorithm twice, the time complexity can be rewritten as $O(I(k, n) + n \log n)$, where $I(k, n)$ is the time complexity of k-BCI problem.

7 Acknowledgment

The authors wish to thank the anonymous referees for their suggestions that help improve the presentation of the paper. One referee's comments lead us to look into the possibility of improving the space complexity of the k-BCI algorithm to $O(n)$.

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilbur, "Geometric Applications of a Matrix-Searching Algorithm," *Algorithmica,* 2 (1987), 195-208.

[2] A. Aggarwal, B. Schieber, and T. Tokuyama, "Finding a Minimum-Weight K-Link Path in Graphs with Monge Property and Applications," *Proc. 9th Symp. on Comput. Geometry,* 1993, 189-197.

[3] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[4] M. Ben-Or, "Lower bounds for algebraic computation trees," *Proc. 15th Annual Symp. on Theory of Computing*, 1983, 80-86.

[5] Z. Drezner, "Competitive location strategies for two facilities," *Regional Science and Urban Economics* **12**, 1982, North-Holland, 485-493.

[6] H. N. Gabow and R. E. Tarjan, "A linear-time algorithm for a special case of disjoint set union," *Proc. 15th ACM Symposium on Theory of Computing* (Boston, Mass, Apr. 25-27), ACM, New York, 1983, 246-251.

[7] S. L. Hakimi, "On locating new facilities in a competitive environment," *ISOLDE II*, June 1981, Skodsborg, Denmark.

[8] W.-L. Hsu and K. H. Tsai, "Linear time algorithms on circular-arc graphs," *Proc. 26th Allerton Conference on Communication, Control and Computing*, 1988, 842-851.

[9] D.T. Lee, "Maximum clique problem of rectangle graphs," In *Advances in Computing Research*, F. P. Preparata, ed., JAI Press (1983), 91-107.

[10] D. T. Lee, M. Sarrafzadeh and Y.F. Wu, "Minimum Cut for Circular-arc Graphs," *SIAM J. Computing.,* 19,6 Dec. 1990, 1041-1050.

[11] D. T. Lee and Y. F. Wu, "Geometric complexity of some location problems," *Algorithmica*, **1**, 1985, 193-211.

[12] A. S. Rao and C. P. Rangan, "Optimal Parallel Algorithms on Circular Arc Graphs," *Info. Proc. Lett.,* **33**, 1989, 147-156.

[13] Y. F. Wu, "Geometric Location and Distance Problems," Ph. D. Dissertation, Dept. of EE/CS, Northwestern University, 1985.