Low complexity algorithms for a consumer partial covering in the plane

Shimon Abravaya* Michael Segal†

October 29, 2007

Abstract. This paper considers a model for locating a consumer within a bounded region in the plane with respect to a set of n existing *pull-push* suppliers. The objective is to maximize the difference of total profits and costs incurred due to the partial covering of the consumer by the suppliers pull and push influence areas. We develop efficient polynomial time algorithms for the resulting problems in the rectilinear and the Euclidean planes where the bounded region is either a rectangle or a constant size polygon, respectively. Based on these solutions, we develop algorithms for evaluating efficiently the objective function at any possible location of the consumer inside the bounded region. We also employ the algorithms for the Euclidean optimization problem and the rectilinear query computation to solve efficiently their corresponding *dynamic* versions, where an appearance of a new supplier or an absence of an existing one occurs. Being easy to implement due to the extensive use of simple data structures, such as the balanced and binary segment tree, and the employment of standard mechanisms, such as the sweep line, the Voronoi diagram and the circular ray shooting, our solutions potentially have wide usability.

Keywords. Operation Research, Computational Geometry, pull-push criteria, semi-obnoxious, partial covering.

1 Introduction

1.1 Model and results

Let S be the set $\{p_1, \ldots, p_n\}$ of pull-push points enclosed in a bounded region $Q \subset \mathbb{R}^2$, such that each point $p_i \in S$ is associated with a positive profit c'_i, a positive cost c''_i, and two positive constants K'_i and K''_i. In the rectilinear model, Q is a rectangle and each

*Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel.

†Department of Communication System Engineering, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel. Partially supported by INTEL and REMON consortium.

point $p_i \in S$ has two positive pull weights w'_i and v'_i, and two positive push weights w''_i and v''_i. For any point $c \in Q$, we denote by $d_x(c, p_i)$ and $d_y(c, p_i)$ the distances between the x coordinates and the distances between the y coordinates, respectively, of c and $p_i, 1 \leq i \leq n$, and define the following two sets:

(*i.*) $S'_c = \{p_i \in S : \max(w'_i d_x(c, p_i), v'_i d_y(c, p_i)) < K'_i\}$,

(*ii.*) $S''_c = \{p_i \in S : \max(w''_i d_x(c, p_i), v''_i d_y(c, p_i)) < K''_i\}$.

In the Euclidean model, Q is a constant size polygon and each point $p_i \in S$ has a positive pull weight w'_i and a positive push weight w''_i. For any point $c \in Q$, we denote by $d(c, p_i)$ the Euclidean distance between c and $p_i, 1 \leq i \leq n$, and define S'_c and S''_c as follows:

(*i.*) $S'_c = \{p_i \in S : w'_i d(c, p_i) < K'_i\}$,

(*ii.*) $S''_c = \{p_i \in S : w''_i d(c, p_i) < K''_i\}$.

In both models, S'_c and S''_c contain each of the pull-push points from S with its pull or push influence area covering c, respectively. Thus, the intersection of these sets may not be empty. The difference of the total profits and costs incurred due to these influences at the location c is given by

$$f(c) = \sum_{p_i \in S'_c} c'_i - \sum_{p_i \in S''_c} c''_i. \tag{1}$$

The following problems hold for both models:

(*i.*) *Optimal coverage.* Find a point $c \in Q$ maximizing f.

(*ii.*) *Dynamic optimal coverage.* Upon an addition of a new pull-push point to S or a removal of an existing one from S, find a point $c \in Q$ maximizing f efficiently (without re-computing the whole solution).

(*iii.*) *Query point computation.* Given $c \in Q$, compute $f(c)$ efficiently.

(*iv.*) *Dynamic query point computation.* Upon an addition of a new pull-push point to S or a removal of an existing one from S, compute $f(c)$, for any given $c \in Q$, efficiently (without re-executing a preprocessing algorithm).

Remark 1 For ease of presentation and to improve this paper clearness and readability, we supply solutions for the above stated problems in the special case where $p_1, \ldots, p_k$ are pull points with push weights and costs that are equal to 0, and $p_{k+1} \ldots p_n$ are push points with pull weights and profits that are equal to 0. We show that each of these solutions, forms, with only slight variations, a similar runtime solution for its equivalent problem in the general case where each $p_i \in S$ is a pull-push point. The precise definitions of a pull, a push and a pull-push point and the discussion about their differences are given at Section 1.2. Remark 2 and Remark 3 define the adjustments required for these solutions generalizations in the rectilinear and Euclidean models, respectively.

We present solutions for the optimal coverage problem in both the rectilinear and Euclidean models, and use the one for the Euclidean case as a preprocessing for the algorithm solving the corresponding dynamic problem. Based on these solutions, we develop algorithms for the query point computation problem in both models, and for its dynamic version in the rectilinear case. The results are summarized at the two theorems bellow.

Theorem 1 *The optimal coverage problem can be solved in $O(n \log n)$ time in the rectilinear model, and in $O(n^2 \log n)$ time in the Euclidean model . Moreover, the algorithm for the Euclidean optimal coverage problem enables, once terminated, solving the Euclidean dynamic optimal coverage problem in $O(n \log n)$ amortized time.*

Theorem 2 *The value of f at any given query point $c \in Q$ can be computed in the rectilinear model in $O(\log n)$ time after an $O(n^2)$ time preprocessing, or in $O(\log^2 n)$ time after an $O(n \log^2 n)$ time preprocessing. Moreover, the second algorithm enables, with slight variations, dynamic computation of the value of f at query points in $O(\log^2 n)$ amortized time. In the Euclidean model, dynamic computation of f can be done in $O(\log n)$ time after an $O(n^3 \log n)$ time preprocessing algorithm is executed.*

1.2 Previous related work

A *semi-obnoxious* supplier is a supplier that incorporates both attractive and noxious aspects. On one hand, it is necessary for the development of its service consumers due to its positive external effects. On the other hand however, it poses some by-product negative external effects to those consumers. These positive and negative external effects are regarded as pulling and pushing forces (also as attracting and repelling forces), and are conflict in the sense that no single location can serve as an optimal one under both forces considerations simultaneously. Each of the consumers is regarded as a *pull-push* element as the supplier is to be located with respect to these already existing consumers. The problem of locating a single semi-obnoxious supplier in the plane with respect to a set of existing pull-push consumers, which employs a maximin, a minimax and a minisum criteria for modeling the pulling and pushing forces, has been extensively acknowledged in the literature [5, 7, 10, 12, 19, 23, 24].

While *full-covering* location problems require that all the customers will be served by the supplier, their *partial-covering* formulations allow neglecting some of the customers. Previous researches deal with the location of pure *desirable* suppliers, where some unserved customers are permitted [8, 11], and with the location of pure *undesirable* suppliers, where the exposure of some customers to the suppliers negative effects is permitted [3, 12, 13, 17, 21]. Plastria [25] supplies an exhaustive overview of continuous pull-push covering location problems, their models and their yielding motivations, with respect to several possible familiar distances including the rectilinear and Euclidean distances.

Ohasawa *et al.* [22] introduce a bicriteria model for locating a semi-obnoxious supplier within a convex polygon, such that a given number of closest and farthest customers are neglected. Thus, by moving out some negatively affected customers and withholding the supplier service from others simultaneously, they lessen opposition of affected minority groups. Gordillo *et al.* [15] solve the problem of locating a semi-obnoxious supplier in the plane where two groups of customers are to be considered. The first is a group of individual customers whose demands must be satisfied by the supplier, and the second is a group of convex polygons areas that have to be protected from the negative external effects of the supplier.

This paper, opposite to previous works, deals with locating a single consumer with respect to already existing semi-obnoxious suppliers, and hence regards each of these suppliers as a pull-push element. Throughout this paper, we regard a supplier that has only positive external effects as a *pull* supplier, a supplier that has only negative external effects as a *push* supplier, and a supplier that may have both these effects as a *pull-push* one. Thus, a pull-push supplier might be only a pull or a push supplier, or both of them simultaneously. The ambivalence reflected by the suppliers pulling and pushing forces yields that the optimal location for the consumer may be a point that can not be served properly by all the suppliers (namely, this point may not be covered by all the suppliers pull influence areas). This, in turn, forms the context in which partial-covering is brought into play in this paper.

Some studies in operations research and computational geometry deal with variants of the optimal coverage problem in both the rectilinear and the Euclidean models. Consider the following related problem: Given a network $G = (V, E)$ with n weighted nodes and m edges of given lengths and given a number $R > 0$, find a point c on the network G which minimizes the number of nodes v such that $w_v dist(c, v) \leq R$. Berman *et al.* [2] supplies an $O(mn \log n)$ time algorithm for this problem. Plastria and Carrizosa [9] consider the problems of placing a supplier in the plane or on a planar network, such that the total number of points lying within the influence radius from the supplier should be minimized. They present $O(\log n)$ time solutions for both problems considering any given radius R, after an initial $O(n^3 \log n)$ time preprocessing. Clearly, our algorithm for the Euclidean optimal coverage problem enables solving in $O(n^2 \log n)$ time their planar problem. Moreover, while the solution in [9] has the advantage of enabling the execution of an interactive procedure over the values of R in real time, our solution supports queries computation and enables solving the corresponding dynamic problem efficiently. Segal [26] supplies an $O(n \log^2 n)$ time and an $O(n^{7/5}\text{polylog}n)$ time algorithms for the special cases of unweighted complete rectilinear and Euclidean networks, respectively, when the supplier must be placed at one of the nodes. We can solve the weighted versions of these problems, for any given general rectilinear or Euclidean network, using our solutions for the rectilinear and Euclidean query point computation problems, respectively. We

first execute the corresponding preprocessing algorithm and then compute the value at each node as a query point. Thus, our solution for the weighted rectilinear version requires $O(n \log^2 n)$ time and for the weighted Euclidean version requires $O(n^3 \log n)$ time. In fact, our query computation algorithms enable solving the discrete versions of the optimal coverage problems on rectilinear and Euclidean networks, when the supplier must be placed at one of the nodes. This, in turn, emphasizes the need for efficient query computation mechanisms.

1.3 Motivations

Our interest in the rectilinear case arises from the physical design process of electronic circuits [18], where an electronic board with several already placed components is given and a new component is to be located. The performance of the circuit is measured by the delay incurred due to the components functioning and the rectilinear connections between them. Thus, locating the new component at a point that is as close as possible to these pull components is required. However, some components produce heat that spread along their connections and may damage the new component. Hence, the location of the new component should be as far as possible from these push components.

Consider a rectangular board R with components $p_1, \ldots, p_n$ already placed, such that the first k are pull components and the last $n-k$ are push components. For each $1 \leq i \leq k$, let α_i be the maximal delay allowed between p_i and the new component, and $1/w_i'$ be the speed of the electric signal that p_i produces. Given a point $c \in R$, we denote by t_1 the number of pull components with $w_i' d(c, p_i) < \alpha_i$. For each $k+1 \leq i \leq n$, let δ_i be the amount of heat that p_i produces, and w_i'' be the linear rate of heat decrease along the rectilinear connections starting at p_i. Let β be the maximal permitted heat at the location of the new component, then for any point $x \in R$, the value $\delta_i - w_i'' d(p_i, x)$ is the amount of heat caused by $p_i, k+1 \leq i \leq n$, at x. Given $c \in R$, we denote by t_2 the number of push components with $\delta_i - w_i'' d(p_i, c) > \beta$. Clearly, the required location for the new component is a point $c \in R$ maximizing $t_1 - t_2$. Transforming the system into L_1 metric distances, we get a special case of the general rectilinear optimal coverage problem with the following data. For each $p_i, 1 \leq i \leq k$, we have $K_i' = \alpha_i$, $v_i' = w_i'$, $c_i' = 1$, $v_i'' = w_i'' = 0$, $c_i'' = 0$, and for each $p_i, k+1 \leq i \leq n$, we have $K_i'' = \delta_i - \beta$, $v_i'' = w_i''$, $c_i'' = 1$, $v_i' = w_i' = 0$, $c_i' = 0$. Note that the rectilinear model is not applicable for this problem as the resulting feasible region (board) is not an axis-parallel rectangle anymore. However, many applications deal with a given huge board with numerous components placed on, and thus the problem must be restricted to a local subset of the given components. In this case, the designers may also omit the given board assuming that the required location, with respect to the sub-problem, lies within the board. Our solution for the rectilinear optimal coverage problem uses the rectangles resulting by the reductions of the components pull and push influence areas to the given rectangle Q. Omitting these

reductions, we get a solution for the rectilinear optimal coverage problem in the plane, which is employable for solving the electronic circuits physical design problem in the above stated case. Alternatively, the solution for the Euclidean optimal coverage problem can be employed by simply considering the rectangular influence areas of the components instead of the circular influence areas considered throughout the Euclidean algorithm, and assigning to Q the resulting feasible region. Obviously, when solving the electronic circuits physical design problem by the second alternative instead of the first one, the running time slightly deteriorates. Note that many of the physical design methods are iterative and hence relocations of some already placed components are possible. Thus, an effective mechanism for computing queries may also be required.

The Euclidean case is strongly motivated by the wide usage of the following two *antennas* systems (e.g., cellular networks). The first system consists of $n-k$ transmitters that produce dangerous electric waves, and k receivers that don't produce such waves and getting closer to them improves the system performance. Thus, each transmitter is considered as push element and each receiver is considered as a pull one. The second system consists of n antennas, such that each serves as a transmitter and a receiver simultaneously, and therefore considered as a pull-push element. Clearly, the Euclidean model can be easily adapted to design and solve a matching optimal coverage problem for each of these two systems. Efficient solutions for the corresponding dynamic problems are most required in case of a new antenna to locate or in case of an antenna fault. A corresponding query point computation mechanism is most useful as the applying object might be a movable one, such as a customer in a cellular network.

This paper is organized as follows. In Section 2 and Section 3 we supply the solutions for the resulting optimal coverage and query point computation problems in the rectilinear and Euclidean models, respectively. Base on these solutions, we also design in Section 2 an algorithm for the dynamic query point computation problem, and in Section 3 an algorithm for the dynamic optimal coverage problem. Finally, we conclude and state some possible future research at Section 4.

2 The rectilinear model

Throughout this paper we assume that the points are in general position, i.e. there are no three collinear points and there are no four coplanar points. Degenerate cases can be handled by symbolic perturbation [14]. The formulation of the problem implies that each pull point $p_i, 1 \leq i \leq k$, defines a pull rectangular region

$$P_i = \left\{ x \in \mathbb{R}^2 | w_i' d_x(x, p_i) < K_i', \quad v_i' d_y(x, p_i) < K_i' \right\}, \tag{2}$$

where c should better reside, and each push point $p_i, k+1 \leq i \leq n$, defines a push rectangular region

$$P_i = \left\{ x \in \mathbb{R}^2 | w_i'' d_x(x, p_i) < K_i'', \quad v_i'' d_y(x, p_i) < K_i'' \right\}, \tag{3}$$

where c better not reside. Let $R_i = P_i \cap Q, 1 \leq i \leq n$, be the rectangles resulting by the reductions of the P_i's to Q. For any point $r \in \mathbb{R}^2$ (alternatively, any segment $r \subset \mathbb{R}^2$) and any rectangle $R_i, 1 \leq i \leq n$, we say that R_i *covers* r if $r \in R_i$ (alternatively, if $r \subset R_i$). We define the *cover profit* of each pull rectangle $R_i, 1 \leq i \leq k$, at r (alternatively, at each point $x \in r$) to be c_i' in case R_i covers r or zero otherwise. Similarly, we define the *cover cost* of each push rectangle $R_i, k+1 \leq i \leq n$, at r (alternatively, at each point $x \in r$) to be c_i'' in case R_i covers r or zero otherwise. We define the *total cover value* of r to be the difference of total cover profits and total cover costs of the pull and the push rectangles, respectively, at r. Denote by $tcv(r)$ the total cover value of r, then

$$tcv(r) = \sum_{\substack{R_i \text{ covers } r \\ 1 \leq i \leq k}} c_i' - \sum_{\substack{R_i \text{ covers } r \\ k+1 \leq i \leq n}} c_i''. \tag{4}$$

Regarding these definitions it is clear that any point $c \in Q$ with the maximal total cover value forms a solution to the optimal coverage problem. Note that in the special case where each point is a pull point associated with a profit of value 1, the required location is a point in Q covered by the maximal number of the resulting pull rectangles, and in the special case where each point is a push point associated with a cost of value 1, the required location is a point in Q covered by the minimal number of the resulting push rectangles.

Bespamyatnikh *et al.* [4] supply an $O(n \log n)$ time algorithm for determining whether a given rectangular area is fully contained in the union of n given rectangles. In Section 2.1 we supply an $O(n \log n)$ time algorithm for the optimal coverage problem based on the solution in [4], and use it to design in Section 2.2 an $O(n^2)$ time preprocessing algorithm which enables the computation of any given query point in $O(\log n)$ time. In Section 2.2 we also supply an $O(n \log^2 n)$ time preprocessing algorithm which enables the computation of query points in $O(\log^2 n)$ time, and extend it to enable such dynamic computation in $O(\log^2 n)$ amortized time.

Remark 2 Consider the general case where each point is a pull-push point. For each $p_i \in S$, we define the resulting pull rectangle R_i' associated with its cover profit c_i' and the resulting push rectangle R_i'' associated with its cover cost c_i'', to be the reductions of the right handsides of (2) and (3) to Q, respectively, as shown above. By the definition of a pull-push point it is clear that one of its resulting pull and push rectangles may be empty due to a possible assignment of the point pull or push weight to zero, respectively, as states Remark 1 (recall that a pull-push point may also be only a pull point or a push

point). Let m the total number of the non-empty rectangles out of $R'_1, R''_1, \ldots, R'_n, R''_n$, then $n \leq m \leq 2n$. Applying each of the algorithms presented in Sections 2.1 and 2.2 on these m non-empty rectangles instead of the R_i's, we obtain a solution with the same time complexity for each of the equivalent general problems. Furthermore, the constants associated with the runtime of these derived algorithms remain significantly small.

2.1 Optimal coverage

Denote by $L = \{x_1, \ldots, x_{2n}\}$ the x coordinates of the endpoints of the horizontal sides of $R_1, \ldots, R_n$ and by $M = \{y_1, \ldots, y_{2n}\}$ the y coordinates of the endpoints of the vertical sides of these rectangles. Assume that each list is sorted in ascending order. The idea is to use a segment tree [20] in iterative fashion. Let T be such a segment tree, and the elements of L be the *events* of T. The leaves of T contain *elementary segments* $[y_j, y_{j+1}), 1 \leq j \leq 2n-1$, in their *range* field. The range at each inner node in T contains the union of the ranges in the nodes of its children. Each node v in T maintains a *total cover profit*, a *total cover cost*, a *total cover value* and a *maximal total cover value in sub-tree*. The first indicates the sum of cover profits of vertical pull rectangle sides that cover the range of v and don't cover the range of its parent, and the second indicates the sum of cover costs of vertical push rectangle sides that cover the range of v and don't cover the range of its parent. The third is simply the total cover value of the range of v. Let T_v be the sub-tree rooted at v, then the fourth field is the maximal total cover value over all the ranges of leaves in T_v, where the rectangles for computing the total cover value of each range are those out of $R_1, \ldots, R_n$ that cover ranges of nodes in T_v and don't cover the range of v. Denote these fields by $tcp(v)$, $tcc(v)$, $tcv(v)$ and $mtcvs(v)$, respectively.

Observation 1 *For any $v \in T$, let A_v be the set of all its ancestors. Then,*

$$tcv(v) = \sum_{u \in A_v \cup \{v\}} tcp(u) - \sum_{u \in A_v \cup \{v\}} tcc(u).$$

For any inner node $v \in T$, let P_v be the set of all paths of the form $v_1, \ldots, v_t$ in T such that v_1 is a son of v, the node $v_i, 1 \leq i \leq t-1$ is the parent of v_{i+1}, and v_t is a leaf.

Observation 2 *For any inner node $v \in T$*

$$mtcvs(v) = \max_{p \in P_v} \left(\sum_{u \in V(p)} tcp(u) - \sum_{u \in V(p)} tcc(u) \right).$$

A vertical line is swept over the plane from left to right stopping at each event of T. At each event x, either a rectangle $R_i, 1 \leq i \leq n$, is added to the rectangles cover or it is deleted from it. Using a recursive scan (starting at the root of T), the vertical side l

of R_i is either inserted to T or deleted from it. Note that l is stored in $O(\log n)$ nodes and is equal to the disjoint union of the ranges of these nodes. Denote by A_i the set of these nodes, then each node $v \in A_i$ is a candidate for the global maximum with the value $tcv(v) + mtcvs(v)$. The update of T by the scan at the event x consists of three phases:

1. The scan "moves" from the root of T towards the nodes in A_i. The scan sums recursively the total cover profits and the total cover costs of the nodes along each path from the root of T to each node in A_i (according to Observation 1).

2. At each node $v \in A_i$:

   (a) If $1 \leq i \leq k$, then the scan update $tcp(v) = tcp(v) \pm c'_i$. Otherwise, $k+1 \leq i \leq n$, and the scan update $tcc(v) = tcc(v) \pm c''_i$. In both cases, the plus holds in case l represents the insertion of R_i, and the minus holds in case l represent the deletion of R_i.

   (b) The scan uses the values calculated at the first phase to compute $tcv(v)$ according to Observation 1. Then, the scan checks if the value $tcv(v) + mtcvs(v)$ is bigger than the current global maximum. If so, the scan updates the current global maximum to be $tcv(v) + mtcvs(v)$.

3. The scan "moves" from each of the nodes in A_i towards the root of T. Let v be the current node that the scan visits, and u, w be its sibling and its parent it T, respectively. The scan updates $mtcvs(w)$ as follows.

$$mtcvs(w) = \max\left(mtcvs(v) + tcp(v) - tcc(v), mtcvs(u) + tcp(u) - tcc(u)\right).$$

It is easy to show that throughout Phase 1 the scan visits $O(\log n)$ nodes, and at Phase 2 it updates $O(\log n)$ nodes. Throughout Phase 3 the scan visits and updates the same nodes visited throughout Phase 1, and for each of these nodes the scan also examines its sibling. Since updating each node requires $O(1)$ time, each of the three phases requires $O(\log n)$ time. Applying this observation at each event $x \in L$, we prove Theorem 1 for the rectilinear case. A closer examination of the scan yields that throughout Phase 1, the scan visits at most $4 \log n$ nodes. This, in turn, yields that the scan visits at most $8 \log n$ nodes in total. Since there are at most $2n$ events $x \in L$, the algorithm checks and updates at most $16n \log n$ nodes of T throughout the whole algorithm. Thus, the constant associated with the $O(n \log n)$ runtime of the algorithm is significantly small, which arises the practical efficiency of this algorithm.

Example 1 Consider the rectilinear optimal coverage problem with the following four points. A pull point p_1 at $(5, 3)$ with $w'_1 = 1/2$, $v'_1 = 1$, $c'_1 = 3$, a pull point p_2 at $(8, 4)$ with $w'_2 = 2/3$, $v'_2 = 1$, $c'_2 = 5$, a push point p_3 at $(6, 6)$ with $w''_3 = 3$, $v''_3 = 3$, $c''_3 = 2$, and a push point p_4 at $(10, 5)$ with $w''_4 = 2$, $v''_4 = 6/5$, $c''_4 = 1$. For $i = 1, 2,$

we have $K'_i = 2$, and for $i = 3, 4$ we have $K''_i = 6$. For ease of presentation we assume that Q can be any rectangle containing these rectangles. Figure 1 shows the rectangles $R_1, \ldots, R_4$ resulting from $p_1, \ldots, p_4$,respectively, and Figure 2 shows the corresponding initial segment tree where $tcp(v)$, $tcc(v)$, $tcv(v)$ and $mtcvs(v)$ are assigned to zeros at each node $v \in T$. Table 1 presents the values of these fields throughout the algorithm. The table's rows and columns correspond to the nodes and the events of T, respectively. The four values at each entry (v, x) are the values of $tcp(v)$, $tcc(v)$, $tcv(v)$ and $mtcvs(v)$, respectively, resulting by the scan occurred upon the event x.

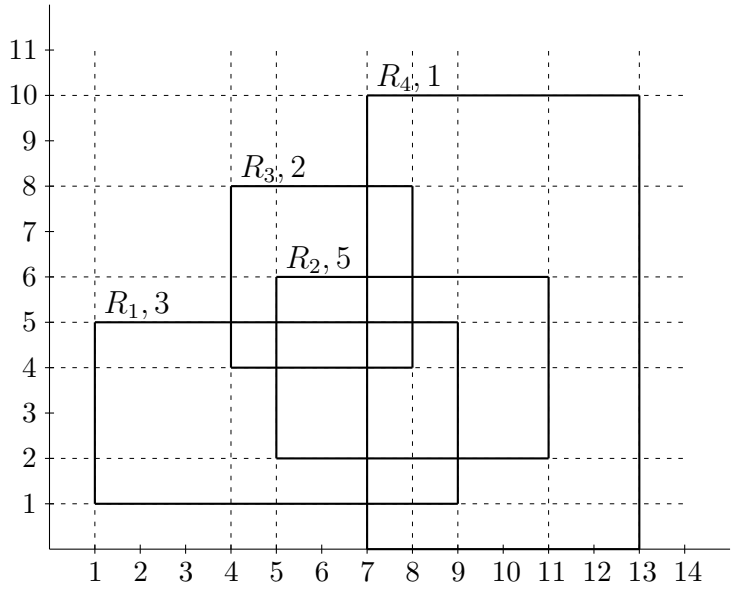

Figure 1: The rectangles $R_1, \ldots, R_4$ resulting from $p_1, \ldots p_4$, respectively.

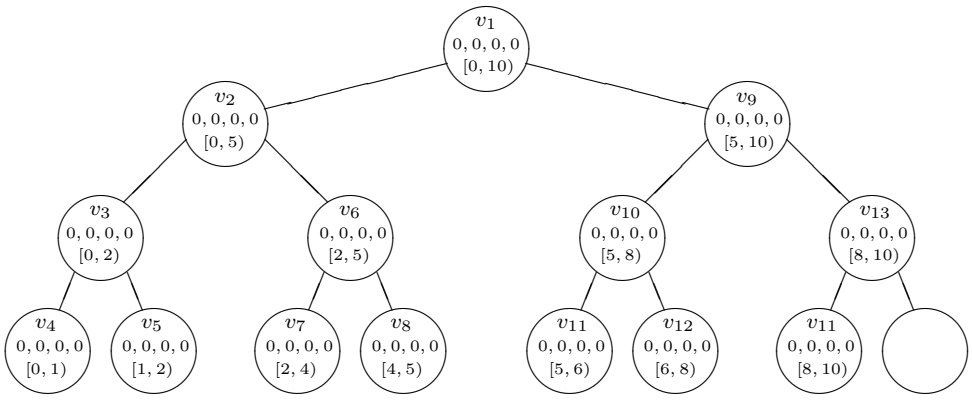

Figure 2: The initial segment tree T

	1	4	5	7	8	9	11	13
v_1	$0,0,0,3$	$0,0,0,3$	$0,0,0,8$	$0,1,-1,8$	$0,1,-1,8$	$0,1,-1,5$	$0,1,-1,0$	$0,0,0,0$
v_2	$0,0,0,3$	$0,0,0,3$	$0,0,0,8$	$0,0,0,8$	$0,0,0,8$	$0,0,0,5$	$0,0,0,0$	$0,0,0,0$
v_3	$0,0,0,3$	$0,0,0,3$	$0,0,0,3$	$0,0,0,3$	$0,0,0,3$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$
v_4	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$
v_5	$3,0,3,0$	$3,0,3,0$	$3,0,3,0$	$3,0,3,0$	$3,0,3,0$	$0,0,-1,0$	$0,0,-1,0$	$0,0,-1,0$
v_6	$3,0,3,0$	$3,0,3,0$	$8,0,8,0$	$8,0,8,0$	$8,0,8,0$	$5,0,4,0$	$0,0,-1,0$	$0,0,-1,0$
v_7	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$
v_8	$0,0,0,0$	$0,2,1,0$	$0,2,1,0$	$0,2,1,0$	$0,0,8,0$	$0,0,8,0$	$0,0,8,0$	$0,0,8,0$
v_9	$0,0,0,0$	$0,0,0,0$	$0,0,0,3$	$0,0,0,3$	$0,0,0,5$	$0,0,0,5$	$0,0,0,0$	$0,0,0,0$
v_{10}	$0,0,0,0$	$0,2,-2,0$	$0,2,-2,5$	$0,2,-2,5$	$0,0,-1,5$	$0,0,-1,5$	$0,0,-1,0$	$0,0,-1,0$
v_{11}	$0,0,0,0$	$0,0,0,0$	$5,0,3,0$	$5,0,3,0$	$5,0,3,0$	$5,0,3,0$	$0,0,-1,0$	$0,0,-1,0$
v_{12}	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$
v_{13}	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$
v_{14}	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$	$0,0,0,0$

Table 1: The values of the fields at each $v \in T$ throughout the optimal coverage algorithm.

2.2 Query point computation

The following $O(n^2)$ time preprocessing algorithm, based on the algorithm for the optimal coverage problem, enables a query point computation in $O(\log n)$ time.

Preprocessing: We construct the $(2n-1) \times (2n-1)$ matrix M, whose rows (skipping row 0) correspond to $[x_i, x_{i+1}), 1 \leq i \leq 2n-1$, and whose columns (skipping column 0) correspond to $[y_j, y_{j+1}), 1 \leq j \leq 2n-1$. Each entry $M_{i,j}$ of M contains the total cover value of $[y_j, y_{j+1})$ at any point in $[x_i, x_{i+1})$. Note that the total cover value of $[y_j, y_{j+1})$ does not change along $[x_i, x_{i+1})$. We initialize each entry of M to 0. Let T be the same segment tree as presented in Section 2.1. At each event x_i, a full scan of T, consists of three phases, is performed. The first two phases are similar to Phase 1 and Phase 2-(a). At the third phase, the scan "moves" from each of the nodes updated at the second phase towards the leaves and keeps on summing the total cover profits and the total cover costs of nodes along each path. When the scan arrives at a leaf v who's range $[y_j, y_{j+1})$, the algorithm updates the entry $M_{i,j}$ to be

$$\sum_{u \in A_v \cup \{v\}} tcp(u) - \sum_{u \in A_v \cup \{v\}} tcc(u),$$

where A_v is the set of all the ancestors of v. Observation 1 confirms that this value is indeed the total cover value of v at any point in $[x_i, x_{i+1})$. Clearly, this scan requires $O(n)$ time which yields that the preprocessing algorithm requires $O(n^2)$ time.

Query computing: For a given point $(x, y) \in Q$, we use a binary search to find the entry $M_{i,j}$ such that $(x, y) \in [x_i, x_{i+1}) \times [y_j, y_{j+1})$ in $O(\log n)$ time. Thus, computing the total cover value of (x, y) requires $O(\log n)$ time.

Example 1 (continued) The following matrix is obtained by running the preprocessing algorithm with the given data:

$$M = \begin{pmatrix} 0 & 0 & 0 & -1 & -1 & -1 & -1 \\ 3 & 3 & 3 & 2 & 2 & -1 & -1 \\ 3 & 3 & 8 & 7 & 7 & 4 & -1 \\ 3 & 1 & 6 & 5 & 7 & 4 & -1 \\ 0 & -2 & 3 & 2 & 4 & 4 & -1 \\ 0 & -2 & -2 & -3 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & -1 & -1 & -1 \end{pmatrix}.$$

We now present an $O(n \log^2 n)$ time preprocessing algorithm which enables a query point computation in $O(\log^2 n)$ time. We define the *life time* of a rectangle $R_i, 1 \leq i \leq n$, to be the horizontal line segment defined by its horizontal side. Each node $v \in T$ maintains a set of life times of rectangles that their vertical side cover the range of v and don't cover the range of its parent.

Preprocessing: The optimal coverage algorithm is performed with the following slight variation. At each insertion event of a rectangle $R_i, 1 \leq i \leq n$, the algorithm stores its life time at each node $w \in A_i$ (indicating that the range of each $w \in A_i$ is covered by the vertical side of R_i at each point along the life time of R_i). Since each $A_i, 1 \leq i \leq n$, contains $O(\log n)$ nodes, at the end of this extension of the optimal coverage algorithm there are $O(n \log n)$ life times of the R_i's stored at the nodes of T. The preprocessing continues by constructing at each node $v \in T$ a segment tree $T_{\text{lt}}(v)$ out of the set of life times stored at v. Let the set of the y-coordinates of the life times at v be the events of $T_{\text{lt}}(v)$. The leaves of $T_{\text{lt}}(v)$ contains elementary segments $[x_i, x_j), 1 \leq i < j \leq 2n$, in their range field, where x_i and x_j are two consecutive x-coordinates of endpoints of life times stored at v. The range of each inner node in $T_{\text{lt}}(v)$ contains the union of the ranges in the nodes of its children. Each node in $T_{\text{lt}}(v)$ maintains the same fields as the nodes of T. The preprocessing continues by executing the optimal coverage algorithm with each $T_{\text{lt}}(v), v \in T$, and the set of life times stored at v. For each node $v \in T$, computing $T_{\text{lt}}(v)$ and executing the optimal coverage algorithm with it requires $O(m \log m)$ time, where m is the number of the life times stored at v. Since the total number of life times stored at the nodes of T is $O(n \log n)$, the preprocessing algorithm requires $O(n \log^2 n)$ time.

Query computing: Given a point $(x, y) \in R$, the algorithm finds the leaf $v \in T$ such that y lies inside its range, and computes the path p_v from the root of T to v in $O(\log n)$ time. Clearly, the set of all life times stored at nodes along p_v is the set of all life times of rectangles who's vertical sides cover the range of v (and therefore cover y) at each point along their life times. For each node w along p_v, the algorithm finds the leaf $u \in T_{\text{lt}}(w)$ such that x lies inside its range, and computes the total cover value of (x, y) with respect

to $T_{\text{lt}}(w)$ in $O(\log n)$ time. Once the algorithm completes these computations, it returns the total of the computed values which is the required $tcv((x, y))$. Thus, computing the total cover value of (x, y) requires $O(\log^2 n)$ time.

The extension required to support the dynamic query point computation employs a dynamic segment tree for T [1] as follows. In the case of an additional point p_{n+1}, inserting the vertical side of the corresponding rectangle R_{n+1} into T and the life time of R_{n+1} into each $T_{\text{lt}}(w), w \in A_{n+1}$, requires $O(\log^2 n)$ amortized time. In the case of an absent point $p_j, 1 \leq j \leq n$, deleting the vertical side of the corresponding rectangle R_j from T and the life time of R_j from each $T_{\text{lt}}(w), w \in A_j$, requires $O(\log^2 n)$ amortized time. In both cases, the query computing remains unchanged and thus requires $O(\log^2 n)$ amortized time.

Example 1 (continued) Once the extended optimal coverage algorithm terminates, v_6 contains the life time of R_1, which is a horizontal line segment with endpoints at $(1, 1)$ and $(9, 1)$, and the life time of R_2, which is a horizontal line segment with endpoints at $(5, 2)$ and $(11, 2)$. Figure 3 shows the corresponding initial segment tree $T_{\text{lt}}(v_6)$. The four zeros at each node $w \in T_{\text{lt}}(v_6)$ indicates the initial values of $tcp(w)$, $tcc(w)$, $tcv(w)$ and $mtcvs(w)$, respectively. Table 2 presents the values of these fields throughout the execution of the preprocessing algorithm with respect to $T_{\text{lt}}(v_6)$ and the two life times.

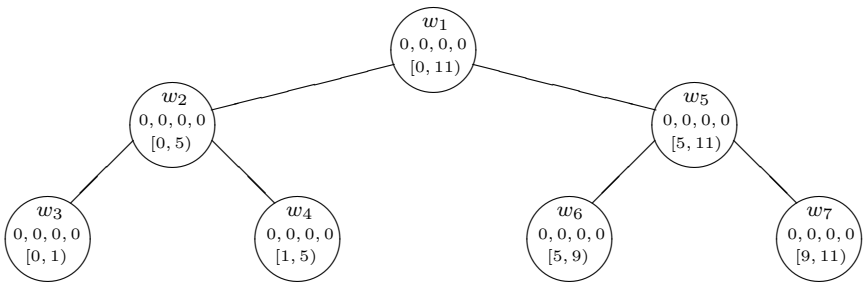

Figure 3: The initial segment tree $T_{\text{lt}}(v_6)$

3 The Euclidean model

As with the rectilinear case, the formulation of the problem implies that each pull point $p_i, 1 \leq i \leq k$, defines a pull circular region

$$C_i = \left\{ x \in \mathbb{R}^2 | w_i' d(x, p_i) < K_i' \right\}, \tag{5}$$

where c should better reside, and each push point $p_i, k+1 \leq i \leq n$, defines a push circular region

$$C_i = \left\{ x \in \mathbb{R}^2 | w_i'' d(x, p_i) < K_i'' \right\}, \tag{6}$$

	R_1 insertion at $y = 1$	R_2 insertion at $y = 2$
w_1	$0, 0, 0, 3$	$0, 0, 0, 8$
w_2	$0, 0, 0, 3$	$0, 0, 0, 3$
w_3	$0, 0, 0, 0$	$0, 0, 0, 0$
w_4	$3, 0, 3, 0$	$3, 0, 3, 0$
w_5	$0, 0, 0, 3$	$5, 0, 5, 3$
w_6	$3, 0, 3, 0$	$3, 0, 3, 0$
w_7	$0, 0, 0, 0$	$0, 0, 0, 0$

Table 2: The values of the fields at each $w \in T_{\text{lt}}(v_6)$ throughout the execution of the optimal coverage algorithm with $T_{\text{lt}}(v_6)$.

where c better not reside. Consider the same definitions of *cover*, *cover profit*, *cover cost* and *total cover value*, as presented in Section 2, with respect to the disks $C_1, \ldots, C_n$ instead of the rectangles $R_1, \ldots, R_n$. The total cover value of any point $r \in \mathbb{R}^2$ (alternatively, any segment $r \subset \mathbb{R}^2$) is given by

$$tcv(r) = \sum_{\substack{C_i \text{ covers } r \\ 1 \le i \le k}} c'_i - \sum_{\substack{C_i \text{ covers } r \\ k+1 \le i \le n}} c''_i. \tag{7}$$

Regarding these definitions it is clear that any point $c \in Q$ with the maximal total cover value forms a solution to the optimal coverage problem. We define a *sub-region* in the plane to be the set of points that are covered by the same set of disks out of the C_i's. One naive solution is to compute all the sub-regions divided by the C_i's and then to compute the total cover value of some point at each sub-region. Solving the problem by computing these sub-regions may be ineffective as there exist up to $O(n^2)$ such sub-regions.

In Section 3.1 we supply the algorithms for the optimal coverage problem and its corresponding dynamic version. In Section 3.2 we supply a preprocessing algorithm, which is based on the algorithm for the optimal coverage problem and enables an efficient computation of any given query point.

Remark 3 Consider the general case where each point is a pull-push point. For each $p_i \in S$, we define the resulting pull disk C'_i associated with its cover profit c'_i and the resulting push disk C''_i associated with its cover cost c''_i, to be the right handsides of (5) and (6), respectively. The statements in Remark 2 considering the disks $C'_1, C''_1, \ldots, C'_n, C''_n$ instead of the rectangles $R'_1, R''_1, \ldots, R'_n, R''_n$, and the algorithms presented in Sections 3.1 and 3.2 instead of those presented in Sections 2.1 and 2.2, respectively, still hold.

3.1 Optimal coverage

Our solution is based on the solution presented in [16] for the following coverage decision problem. Let $S = \{s_1, \ldots, s_n\}$ be a set of n sensors and A be a two dimensional area.

Each sensor $s_i \in S$ is located inside A and has a sensing range of r_i (namely, s_i can monitor any point that is within a distance of r_i from its location). Given a natural number m, the problem is to determine whether every point in A is covered by at least m sensors. In order to solve the problem, Huang *et al.* [16] supply an $O(n \log n)$ time algorithm that for a given sensor and a given integer j, determines whether every point on the sensor's sensing range perimeter is covered by at least j other sensors sensing ranges. They first present an algorithm for the special case of identical r_i's and later extend it for the general case of different r_i's. In both cases, the algorithm computes all the intersection points of the sensor's (sensing range) perimeter and the perimeters of all the other relevant sensors (sensing ranges). The algorithm places these points on the line segment $[0, 2\pi]$ and then sorts them in an ascending order. Finally, the algorithm traverses the line segment from left to right and determines the perimeter cover of the sensor. The following lemma shows how to compute the total cover value of points lying inside the sub-regions attached to a given arc, when the total cover value of this arc is known.

Lemma 1 *Consider any arc r, contained in the perimeter of the disk C_i, that divides a sub-region in the plane into two sub-regions. Denote by D_1 the sub-region that is outside C_i, and by D_2 the sub-region that is inside C_i. For any point $x \in D_1 \cup D_2$, we have*

$$tcv(x) = \begin{cases} tcv(r), & 1 \le i \le n, \quad x \in D_1 \\ tcv(r) + c'_i, & 1 \le i \le k, \quad x \in D_2 \\ tcv(r) - c''_i, & k+1 \le i \le n, \quad x \in D_2. \end{cases}$$

Example 2 Consider the Euclidean optimal coverage problem with the polygon Q defined by the breakpoints $(0.5, 2), (1.5, 4), (5, 4), (5, 3), (6, 2)$, and the following four points. A pull point p_1 at $(3, 2)$ with $w'_1 = 1/2$ and $c'_1 = 3$, a pull point p_2 at $(1.75, 2.75)$ with $w'_2 = 1/2$ and $c'_2 = 5$, a push point p_3 at $(3, 3.5)$ with $w''_3 = 1$ and $c''_3 = 2$, and a push point p_4 at $(4, 2.5)$ with $w''_4 = 1$ and $c''_4 = 1$. For $i = 1, 2$ we have $K'_i = 1$, and for $i = 3, 4$ we have $K''_i = 2$. Figure 4 shows the disks $C_1, \ldots, C_4$ resulting from $p_1, \ldots, p_4$,respectively. Let r be the arc of the perimeter of C_1 with x_4 and x_5 as left and right (clockwise) endpoints. Since only C_4 covers r, we have $tcv(r) = -c''_4 = -1$. Consider the regions D_1 and D_2 touching r such that D_1 is outside C_1 and D_2 is inside C_1, as shown in Figure 4. Any point $x \in D_1$ is covered by the push disk C_4 alone and thus $tcv(x) = tcv(r) = -c''_4 = -1$. Any point $y \in D_2$ is covered by both the pull disk C_1 and the push disk C_4. Thus, $tcv(y) = c'_1 - c''_4 = c'_1 + tcv(r) = 3 - 1 = 2$.

Let D be any of the sub-regions divided by the C_i's. By the definition of a sub-region it is clear that the total cover value of any point in D can be computed as shown in Lemma 1 using any of the arcs comprising its boundary.

Example 2 (continued) Let α be the intersection point of the perimeters of C_3 and C_4

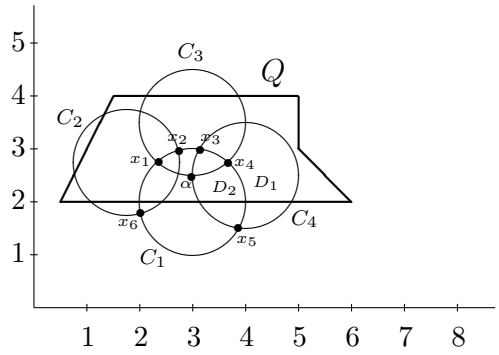

Figure 4: The disks $C_1, \ldots, C_4$ resulting from $p_1, \ldots, p_4$, respectively.

that lies inside C_1. Denote by r_1, r_3, r_4 the arcs comprising the boundary of D_2 such that, r_1 is the arc of the perimeter of C_1 with endpoints x_4 and x_5, r_3 is the arc of the perimeter of C_3 with endpoints α and x_4, and r_4 is the arc of the perimeter of C_4 with endpoints x_5 and α. Note that $tcv(r_1) = -c_4'' = -1$, $tcv(r_3) = c_1' - c_4'' = 2$ and $tcv(r_4) = c_1' = 3$. Clearly, for any point $x \in D_2$, we have $tcv(x) = c_1' - c_4'' = 2$. Using the arcs comprising the boundary of D_2, this value may also be obtained by each of the followings: (i) D_2 is inside C_1 and hence $tcv(x) = tcv(r_1) + c_1' = -1 + 3 = 2$, (ii) D_2 is outside C_3 and hence $tcv(x) = tcv(r_3) = 2$, and (iii) D_2 is inside C_4 and hence $tcv(x) = tcv(r_4) - c_4'' = 3 - 1 = 2$.

The above stated lemma and the conclusion it yields imply that in order to find a point in the plane with a maximal total cover value, it suffices to compute the total cover values of arcs comprising the boundaries of the sub-regions. Moreover, in order to find such point in Q, it suffices to compute the total cover values of only those arcs that also intersect Q. Below we supply a solution for the special case of identical size C_i's. The solution for the general case of different size C_i's is obtained by the same extension as presented in [16, Section 3.2] for the general case of different r_i's, and has the same time complexity.

At the preprocessing phase we construct the data structure for circular ray shooting over the set of all the edges of Q in $O(|Q| \log |Q|)$ time [6], where $|Q|$ is the size of this set. We use this data structure throughout the algorithm to decide in $O(\log^2 |Q|)$ time whether a given arc r intersects Q by circular ray shooting query, which in turn yields that the sub-regions touching r intersect Q. We perform a circular ray shooting query to find if r intersects some edge of Q in $O(\log^2 |Q|)$ time. If so, $r \cap Q \neq \phi$. Otherwise, we decide as explained below whether at least one of the endpoints of r lies inside of Q. If so, then $r \subset Q$. Otherwise, $r \cap Q = \phi$. For each $C_i, 1 \leq i \leq n$, the following algorithm finds the sub-region with the maximal total cover value (namely, every point in it has the maximal total cover value) over the set of all sub-regions touching the perimeter of C_i that intersect Q.

1. Compute the intersection points of the perimeter of C_i and the perimeters of all the other disks as presented in [16, Section 3.1]. Place these points on the line segment $[0, 2\pi]$ and then sort them in an ascending order into a list L. Mark each point as a left or right boundary of coverage range.

2. Traverse the line segment $[0, 2\pi]$ by visiting each element in the sorted list L from left to right. Denote by x the current element of L, by r_{prev} the arc of the perimeter of C_i defined by the previous element of L and x, by r_{next} the arc of the perimeter of C_i defined by x and the next element of L, and by $C_j, j \neq i$, the disk that its perimeter intersect the perimeter of C_i at x. Compute $tcv(r_{\text{next}})$ by

$$tcv(r_{\text{next}}) = \begin{cases} tcv(r_{\text{next}}) = tcv(r_{\text{prev}}) + c'_j, & 1 \leq j \leq k,\ x \text{ is a left point} \\ tcv(r_{\text{next}}) = tcv(r_{\text{prev}}) - c'_j, & 1 \leq j \leq k,\ x \text{ is a right point} \\ tcv(r_{\text{next}}) = tcv(r_{\text{prev}}) - c''_j, & k+1 \leq j \leq n,\ x \text{ is a left point} \\ tcv(r_{\text{next}}) = tcv(r_{\text{prev}}) + c''_j, & k+1 \leq j \leq n,\ x \text{ is a right point.} \end{cases}$$

   If $r_{\text{next}} \cap Q = \phi$, continue with the next element of L (go to Phase 2). Otherwise, the two sub-regions attached with r_{next} are candidates for the optimal coverage location. Hence,

   (a) If $1 \leq i \leq k$, then the sub-region that is inside C_i is a candidate with the value $tcv(r_{\text{next}}) + c'_i$. Compare it with the global maximum and return a location pointer accordingly.

   (b) If $k+1 \leq i \leq n$, then the sub-region that is outside C_i is a candidate with the value $tcv(r_{\text{next}})$. Compare it with the global maximum and return a location pointer accordingly.

As we already mentioned, using the algorithm for a circular ray shooting presented in [6], any polygon of size m can be preprocessed in $O(m \log m)$ time such that a circular ray shooting query computation requires $O(\log^2 m)$ time. Boland *et al.* [6] also show that the constants associated with the runtimes of the preprocessing and the query computation are equal and smaller than 5. We denote both these constants by α_0. For any polygon Q of constant size m and any point $p \in \mathbb{R}^2$, deciding if $p \in Q$ is done by simply drawing a horizontal line, starting from p, and checking whether the number of edges of Q intersecting this line is odd. This requires m computational steps. Notice, however, that this bound can be improved to $O(\log m)$ for a convex polygon of size m by the binary search slicing procedure. We present the analysis below for the constant size convex polygon.

Assuming Q is a polygon of size m, the preprocessing phase requires $O(m \log m)$ time, and checking whether any given arc intersects Q requires $O\left(\max(\log^2 m, \log m)\right) = O(\log^2 m)$ time. Moreover, the constants associated with runtimes of these computations

are α_0 and $\alpha_1 = \max(\alpha_0, 2)$, respectively, and both are smaller than 5. The computation of the intersection points of any two disks requires a single comparison of the disks in general. Thus, the computation of the set of all the intersection points of each disk with all the other disks, requires less than n such comparisons. For each disk, the size of this set is at most $2n$, and hence computing and sorting this set at Phase 1 requires $O(n \log n)$ time, where the constant associated with this runtime dependant solely on that of the sorting algorithm. Let α_2 be the constant associated with the runtime of an efficient sorting algorithm such as the merge sort or the heap sort. Employing this sorting algorithm, Phase 1 requires $O(n \log n)$ time and the constant associated with its runtime is α_2. For each disk $C_i, 1 \leq i \leq n$, the size of L is at most $2n$ and thus the number of the arcs examined at Phase 2 is at most $2n$. Applying the above complexity considerations, we conclude that Phase 2 requires $O(n \log^2 m)$ time where the constant associated with its runtime is $2\alpha_1 < 10$. This yields that the above algorithm requires $O\left(n \max\left(\log n, \log^2 m\right)\right)$ time and the constant associated with its runtime is given by $\max(2\alpha_1, \alpha_2)$. Thus, our solution for the Euclidean optimal coverage problem requires $O\left(\max\left(m \log m, n^2 \max(\log n, \log^2 m)\right)\right)$ time and the constant associated with its runtime is given by $\max(2\alpha_1, \alpha_2)$. This proves Theorem 1 for the Euclidean case and shows the efficiency of our solution in practice assuming Q is a constant size polygon. For ease of presentation and to improve the paper's readability, we regard this case only throughout the rest of this paper and hence omit any complexity consideration that may arise due to the size of Q, i.e., $\log^2 m \in O(1)$.

Example 2 (continued) Figure 5 shows the interval $[0, 2\pi]$ resulting by the execution of the above algorithm with C_1 (starting from the leftmost point of C_1). Denote by $r_1, \ldots, r_6$ the arcs of C_1 defined by the pairs of successive (clockwise) intersection points (x_1, x_2), (x_2, x_3), (x_3, x_4), (x_4, x_5), (x_5, x_6) and (x_6, x_1), respectively. Each $r_j, 1 \leq j \leq 6$, is associated with two numbers. The first number is $tcv(r_j)$ and the second number is its value as a candidate for the global maximum. However, the algorithm does not consider r_5 for the global maximum computation as $r_5 \cap Q = \phi$. Hence, the second value associated with r_5 is "none".

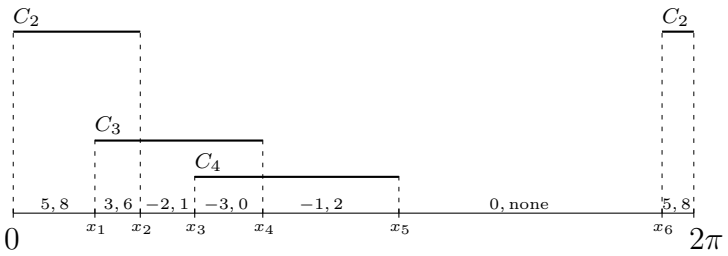

Figure 5: The interval $[0, 2\pi]$ associated with C_1.

To solve the dynamic optimal coverage problem we consider two cases:

i. An additional point p_{n+1}. Applying the above algorithm on C_{n+1}, we get the sub-region with the maximal total cover value touching the perimeter of C_{n+1} over the set of all the sub-region that intersect Q. By the algorithm execution we also achieve the intersection points (if any) of C_{n+1} and each of the C_i's. We apply the above algorithm on each $C_i, 1 \leq i \leq n$, with the following slight variation. At the first phase we just perform a sorted insertion of the intersection points of C_i and C_{n+1} into the list L, and hence L remain sorted. The second phase remains the same.

ii. An absent point $p_j, 1 \leq j \leq n$. We apply the above algorithm on each $C_i, i \neq j$ with the following slight variation. At the first phase we just remove the intersection points of C_i and C_j from L, and hence L remain sorted. The second phase remains the same.

Since an insertion to a sorted L and a removal from a sorted L both require linear time, the algorithm for the dynamic problem takes $O(n^2)$ time. In order to improve this time complexity, we change the optimal coverage algorithm as follows. For each $C_i, 1 \leq i \leq n$, we compute at the first phase the set of arcs defined by the two intersection points of C_i and $C_j, j \neq i$, and then place their corresponding straight line segments (the straightened arcs) on the line segment $[0, 2\pi]$ as shown in Figure 5. At the second phase we construct a dynamic segment tree T_i out of the set of these line segments [1], and execute the algorithm for the rectilinear optimal coverage problem with T_i in a similar way we did for each $T_{\text{lt}}(v), v \in T$, in Section 2.2. Actually we execute an extension of this algorithm which considers only those nodes in T_i that the arcs defining their ranges intersect Q. By that we obtain a point in $[0, 2\pi]$ with the maximal total cover value. This completes the preprocessing algorithm which requires $O(n^2 \log n)$ amortized time.

Consider the case of an additional point p_{n+1}. We apply the improved optimal coverage algorithm with C_{n+1} and get the sub-region with the maximal total cover value touching the perimeter of C_{n+1} in $O(n \log n)$ amortized time. For each $1 \leq i \leq n$, denote by l_i be the straight line segment resulting by straightening the arc defined by the two intersection points of C_i and C_{n+1}. Inserting l_i into T_i we obtain the sub-region with the maximal total cover value touching the perimeter of C_i in $O(\log n)$ amortized time. Applying this observation to each $C_i, 1 \leq i \leq n$, that intersects C_{n+1}, we compute the global maximum in $O(n \log n)$ amortized time.

Consider the case of an absent point $p_j, 1 \leq j \leq n$. By deleting from each $T_i, i \neq j$, the straight line segment corresponding to the straightened arc defined by the intersection points of C_i and C_j, and omitting the result obtained at T_j, we get the global maximum in $O(n \log n)$ amortized time. This completes the proof for Theorem 1.

3.2 Query point computation

Let V be the set of the intersection points of the perimeters of the C_i's, and E be the set of arcs of the C_i's defined by these intersection points. Clearly, V contains the vertices of the sub-regions divided by the C_i's, and E contains the arcs comprising the boundaries of these sub-regions. By the algorithm for the optimal coverage problem we get these two sets such that each arc in E is associated with its total cover value. We use V and E to design an $O(n^3 \log n)$ time preprocessing that enables computing queries in $O(\log n)$ time as follows.

Preprocessing: For each arc $r \in E$, we define l to be the straight line connecting its endpoints, and L to be the set of all these lines. To each point $v \in V$, we attach all the arcs and lines from $E \cup L$ such that v forms one of their endpoints. For each point $v \in V$, we sort all the lines attached to v by their angles in an ascending order. Finally, we construct the Voronoi diagram VD of V and the corresponding point location data structure [1].

Query computing: Given a point $p \in Q$, we find the cell of VD containing p using the point location data structure. Let $v \in V$ be the intersection point defining this cell. We first compute the straight line l connecting p and v, and then use a binary search to find the two lines $l_1, l_2 \in L$, such that l lies in between them. Let $r_1, r_2 \in E$ be the arcs defining l_1 and l_2, respectively. We use $tcv(r_1)$ and $tcv(r_2)$ to compute $tcv(p)$ according to Lemma 1.

For each point $v \in V$, sorting its $O(n)$ attached lines requires $O(n \log n)$ time. Constructing the Voronoi diagram VD of V and the corresponding point location data structure requires $O(n^2 \log n)$ time. Thus, the preprocessing requires $O(n^3 \log n)$ time. Note that finding the cell of VD containing any given query point requires $O(\log n)$ time.

Example 2 (continued) Let β be the intersection point of the perimeters of C_3 and C_4 that lies outside C_1, r_2 be the arc of C_1 with x_5 and x_6 as left and right (clockwise) endpoints, and r_5 be the arc of C_4 with β and x_5 as left and right (clockwise) endpoints. Denote by l_1, l_2, l_4 and l_5 the lines defined by the endpoints of r_1, r_2, r_4 and r_5, respectively, as shown in Figure 6. Consider a query point q located at $(x, y) \in Q$, such that the Euclidean distance between q and x_5 is smaller than all the Euclidean distances between any of the intersection points of the perimeters of $C_1, \ldots, C_4$ and q. Thus, the cell of VD containing q is the one defined by x_5. Denote by l the line crossing q and x_5. Using a binary search the algorithm computes the two lines l_1 and l_4 such that l lies in between them. This, in turn, yields that the point q lies inside the sub-region bounded by the arcs r_1 and r_4. By Lemma 1, the total cover value of q is 2.

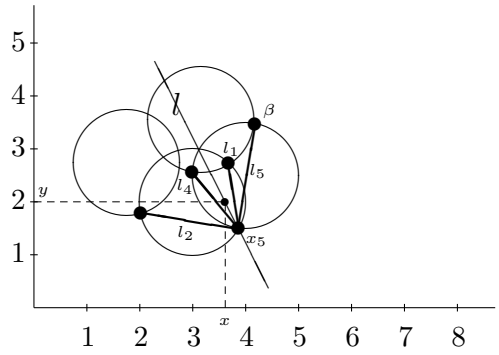

Figure 6: Euclidean query point computation

4 Conclusions

In this paper we have presented a variety of problems dealing with cover value of regions, and finding an optimal coverage location with respect to region partitions produced by a given objective function. Our algorithms can be generalized to deal with dynamic data sets and are able to compute solutions for given queries. One of promising future directions is to apply our techniques to different objective functions with outliers, i.e. taking into consideration a selective group of representative objects. Another possible research can deal with improving the time complexity of dynamic solutions.

References

[1] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, *Computational Geometry: Algorithms and applications*, Springer-Verlag, Berlin (1997).

[2] O. Berman, Z. Drezner and G. Wesolowsky, Minimum covering criterion for obnoxious facility location on a network, *Networks* 28 (1996), 1–5.

[3] O. Berman, Z. Drezner and G. Wesolowsky, The expropriation location problem, *Journal of the Operational Research Society* 54 (2003), 769–776.

[4] S. Bespamyatnikh, K. Kedem, M. Segal and A. Tamir, Optimal facility location under various distance functions, *International Journal of Computational Geometry and Applications* 10 (2000), 523–534.

[5] R. Blanquero and E. Carrizosa, A D.C. Biobjective location model, *Journal of Global Optimization* 23 (2002), 139-154.

[6] R. P. Boland and J. Urrutia, A simpler circular ray shooting algorithm, *13th Canadian Conference on Computational Geometry* (2001), 37–40.

[7] J. Brimberg and H. Juel, A bicriteria model for locating a semi-desirable facility in the plane, *European Journal of Operational Research* 106 (1998), 144–151.

[8] J. Brimberg and C. ReVelle, A multi-facility location model with partial satisfaction of demand, *Studies in Locational Analysis* 13 (1999), 91-101.

[9] E. Carrizosa and F. Plastria, Undesirable facility location with minimal covering objectives, *European Journal of Operational Research* 119 (1999), 158–180.

[10] E. Carrizosa, E.Conde and D. Romero Morales, Semi-Obnoxious location models: A global optimization approach, *European Journal of Operational Research* 102 (1997), 295–301.

[11] E. Carrizosa and F. Plastria, Polynomial algorithms for parametric minquantile and maxcovering planar location problems with locational constraints, *TOP* 6 (1998), 179-194.

[12] E. Carrizosa and F. Plastria, Location of Semi-Obnoxious Facilities, *Studies in Locational Analysis* 12 (1999), 1-27.

[13] Z. Drezner and G.Wesolowsky, Finding the circle or rectangle containing the minimum weight of points, *Location Science* 2 (1994), 83-90.

[14] H. Edelsbrunner and E. P. Mücke, Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms, *ACM Transactions on Graphics*, volume 9, issue 1, (1990), 66–104.

[15] J. Gordillo, F. Plastria and E. Carrizosa, Locating a Semi-Obnoxious Facility With Repelling Polygonal Regions, *Euro Winter Institute on Location and Logistics Book*, Estoril (Portugal), Ana Paias, Francisco Saldanha da Gama Editors. (2007), 166–182.

[16] C. Huang and Y. Tseng, The coverage problem in a wireless sensor network, *Mobile Networks and Applications* 10 (2005), 519–528.

[17] M. Katz, K. Kedem and M. Segal, Improved algorithms for placing undesirable facilities, *Computer and Operation Research* 29 (2002), 1859–1872.

[18] M. Lorenzetti and B. Preas, *Physical Design Automation of VLSI Systems*, Benjamin/Cummings Publishing Company, California (1988).

[19] E. Melachrinoudis and Z. Xanthopulos, Semi-obnoxious single facility location in Euclidean space, *Computers and Operations Research* 30 (2003), 2191–2209.

[20] K. Mehlhorn, *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, Springer-Verlag (1984).

[21] J. Muñoz-Pérez and J. J. Saameño-Rodríguez, Location of an undesirable facility in a polygonal region with forbidden zones, *European Journal of Operational Research* 114 (1999), 372-379.

[22] Y. Ohsawa, F. Plastria, and K. Tamura, Euclidean Push-Pull Partial Covering Problems, *Computers and Operations Research* 33 (2006), 3566-3582.

[23] Y. Ohsawa, Bicriteria Euclidean location associated with maximin and minimax criteria, *Naval Research Logistics* 47 (2000), 581–592.

[24] Y. Ohsawa and K. Tamura, Efficient Location for a Semi-Obnoxious Facility, *Annals of Operations Research* 123 (2003), 88–173.

[25] F. Plastria, Continuous covering location problems, in H. Hamacher and Z. Drezner, *Location analysis: theory and applications* , Springer, New-York (2001), 39–83.

[26] M. Segal, Placing an obnoxious facility in geometric networks, *Nordic Journal of Computing* 10 (2003), 224–237.