On the Coverage of Points in the Plane by Disks Centered at a Line

Logan Pedersen* Haitao Wang†

Abstract

Given a set P of n points and a line L in the plane, we consider the problem of computing a set S of disks centered at L such that their union covers all points of P. The cost of a disk is defined as a function $f(r) = r^\alpha$, where $\alpha \geq 1$ is a constant and r is the radius of the disk. The objective is to minimize the total sum of the cost of all disks of S. Previously [Alt et al., SoCG 2006], the problem was solved in $O(n^4 \log n)$ time in any fixed L_p metric (and in $O(n^2 \log n)$ time if $\alpha = 1$). In this paper, we present a new algorithm that runs in $O(n^2)$ time for any $\alpha \geq 1$ in any fixed L_p metric. In addition, we also give algorithms for two variations of the 1D problem where all points of P are in L: (1) there is an upper bound k on $|S|$, and (2) the disk centers must be chosen from another given set of potential locations on L.

1 Introduction

In this paper, we consider the following disk coverage problem. Given a set P of n points and a line L in the plane, we want to find a set S of disks centered at L such that each point of P is covered by at least one disk and the total sum of the cost of all disks of S is minimized. Here, the *cost* of a disk of S is defined to be $f(r) = r^\alpha$ for a given constant $\alpha \geq 1$, where r is the radius of the disk. Note that if $\alpha = 1$ (resp., $\alpha = 2$), we are minimizing the total sum of the radii (resp., areas) of all disks. The problem is motivated by power consumption models in wireless network design, where α is often larger than or equal to 2 [3, 13, 21]. We consider the general metric L_p for any $p \geq 1$, where a point q is said to be covered by a disk centered at a point c with radius r if the L_p distance between q and c is at most r. We refer to the problem as the *disk coverage* problem.

Previously, Alt et al. [3] gave an algorithm that solves the problem in $O(n^4 \log n)$ time in any L_p metric and for any $\alpha \geq 1$. Better algorithms for some special cases of the problem were also presented in [3]. If $\alpha = 1$, then they solved the problem in $O(n^2 \log n)$ time in any L_p metric. In the L_∞ metric, they gave an $O(n^3 \log n)$ time algorithm for any $\alpha \geq 1$.

*Department of Computer Science, Utah State University, Logan, UT 84341, USA, logan.pedersen@aggiemail.usu.edu

†Department of Computer Science, Utah State University, Logan, UT 84341, USA, haitao.wang@usu.edu

In this paper, we propose a new algorithm of $O(n^2)$ time for any L_p metric and any $\alpha \geq 1$, which improves the $O(n^4 \log n)$ time algorithm in [3] by a more than quadratic factor. Our algorithm first reduces the problem to finding a shortest path in a directly acyclic graph (DAG) G, with $n + 1$ vertices and $\Theta(n^2)$ edges. One difficulty is to compute the weights of the edges of G. We propose an algorithm that can compute each edge weight in $O(1)$ amortized time. Consequently, a shortest path in G can be found in $O(n^2)$ time by a textbook dynamic programming algorithm [12].

In addition, we consider the one-dimensional version of the problem where the points of P are all on L (in contrast, we may consider the above more general problem as a "1.5D" problem). Note that if there are no constraints on the disks, then one could obtain an optimal solution by placing a disk with zero radius at each point of P. Thus, we consider two variations with constraints on the disks.

In the first problem, we are allowed to place at most k disks, for a given $k \in [1, n]$. To our best knowledge, we have not seen any previous work on this problem before. We reduce the problem to computing a k-link shortest path in a DAG of $n + 1$ vertices and $O(n^2)$ edges, which can then be solved in $O(kn^2)$ time by an easy dynamic programming algorithm. Further, we show that the edge weights of the graph obey the concave Monge condition, and consequently, we can solve it in $O(nk)$ [1], $O(n\sqrt{k \log n} + n \log n)$ [2], or $n2^{O(\sqrt{\log k \log \log n})}$ time [22], after the points of P are sorted on L. We refer to this problem as the *k-interval coverage* problem because a disk in 1D becomes an interval on L.

In the second problem, in addition to P, we are given another set Q of m points on L as the potential locations for the centers of the disks (i.e., the center of each disk of S must be in Q). This problem has been studied before. Bilò et al. [6] first showed that the problem is solvable in polynomial time. Lev-Tov and Peleg [17] gave an algorithm of $O((n + m)^3)$ time for any $\alpha \geq 1$. Some progress has been made recently by Biniaz et al. [7], who proposed an $O((n + m)^2)$ time algorithm for the case $\alpha = 1$. In this paper, we solve the problem in $O(n(n + m) + m \log m)$ time for any $\alpha \geq 1$, again by reducing it to finding a shortest path in a DAG. We refer to this problem as the *discrete interval coverage* problem.

1.1 Related Work

Some faster approximation algorithms are also known for these problems. For the discrete interval coverage problem, Lev-Tov and Peleg [17] derived a linear time algorithm with approximation ratio 4 for the case $\alpha = 1$, and the ratio was reduced to 3 by Alt et al. [3] with the same running time. Alt et al. [3] also proposed a 2-approximation algorithm with $O(m + n \log m)$ time for the case $\alpha = 1$. Efficient approximation algorithms were also given by Alt et al. [3] for the 1.5D disk coverage problem in the L_∞ metric for any $\alpha \geq 1$. In addition, Alt et al. [3] considered a variation of the 1.5D disk coverage problem where we are given the slope of L but its location may be chosen freely to minimize the total cost. The problem was shown not computable by radicals when $\alpha = 1$, but FPTAS were given for $\alpha = 1$ and $\alpha > 1$ [3].

The discrete case of the 2D disk coverage problem where both P and Q are points in the Euclidean plane is shown to be NP-hard for any $\alpha > 1$ [3]. For the case $\alpha = 1$, Lev-Tov and Peleg [17] gave a PTAS, and later, Gibson et al. [14] showed that the problem is solvable in polynomial time[1]. A variant of the problem in which $P = Q$ but there is an upper bound k on the number of disks is also solved in polynomial time [14]. Other variations of the problem are considered elsewhere, e.g., [5, 6]

The traditional k-center and k-median problems are closely related to the disk coverage problem. Roughly speaking, the k-center problem is to minimize the largest radius of the disks and the k-median problem is to minimize the total sum of distances from all points to their closest disk centers. Both problems have an upper bound k on the number of disks that can be used. These problems are in general NP-hard [19], but have polynomial time solutions in some special cases, e.g., the 1D case [4, 10, 11, 15, 20], the 1.5D case [8, 16, 23], 1 or 2-center in the Euclidean plane [9, 18], etc.

Paper Outline. The rest of the paper is organized as follows. Section 2 defines some notation. In Section 3, we present our algorithm for the 1.5D disk coverage problem. The algorithms for the 1D problems are given in Section 4. Section 5 concludes the paper with remarks on possible extensions of our results to other more general cost functions $f(r)$ and possible improvements on our results.

2 Preliminaries

For ease of exposition, for all three problems studied in the paper, we make a general position assumption that no two points of P have the same x-coordinate. Without loss of generality, we assume that the line L is the x-axis. These assumptions can be easily lifted without affecting the performance of our algorithms.

In each problem, we first sort all points of P by their x-coordinates from left to right, and let the sorted list be $p_1, p_2, \ldots p_n$. For any i, j with $1 \leq i \leq j \leq n$, let $P[i, j]$ denote the sublist $p_i, p_{i+1}, \ldots, p_j$.

For any point q in the plane, let $x(q)$ and $y(q)$ denote the x- and y-coordinates of q, respectively.

For any two points q and q' in the plane, we use $d_p(q, q')$ to denote their L_p distance. We say that q is *to the left* of q' if $x(q) \leq x(q')$, and q is *to the right* of q' if $x(q) \geq x(q')$.

In any solution of each problem, if a point p_i is covered by a disk centered at c, then we call c a *server* and we say that p_i is "served" or "covered" by c.

3 The 1.5D Disk Coverage Problem

In this section, we present our $O(n^2)$ time algorithm for the 1.5D disk coverage problem. In this problem, the points of P are in the plane. Recall that L is the x-axis.

We assume that all points of P are above or on the x-axis (since otherwise if a point $p \in P$ was below the x-axis, we could replace p by its symmetrical point with the x-axis without affecting the optimal solution).

Recall that $P = \{p_1, p_2, \ldots p_n\}$, already sorted on L from left to right. We first model the problem to a shortest path problem in a directly acyclic graph (DAG) G. To this end, the following lemma is critical (the lemma is also applicable to the two 1D problems).

Lemma 1 *In any fixed L_p metric, for any $\alpha \geq 1$, there exists an optimal solution in which the points of P served by the same server are consecutive in their index order.*

Proof. Consider an optimal solution in which the lemma statement does not hold. Then, there must exist two consecutive points p_i and p_{i+1} (we call them a *conflict pair*) and two servers c_1 and c_2 with $x(c_1) < x(c_2)$ such that p_i is served by c_2 and p_{i+1} is served by c_1 in the solution (e.g., see Fig. 1). In the following, we show that we can switch the service of p_i and/or p_{i+1} so that either they are served by the same server, or we can use c_1 to serve p_i and use c_2 to serve p_{i+1}, without affecting the total cost of the solution.

Let r_1 be the radius of the disk centered at c_1, and r_2 the radius of the disk centered at c_2. Since p_{i+1} is served by c_1 and p_i is served by c_2, we have $d_p(c_1, p_{i+1}) \leq r_1$ and $d_p(c_2, p_i) \leq r_2$. Without loss of generality, we assume that $y(p_i) \geq y(p_{i+1})$. Depending on whether $x(p_{i+1}) \leq x(c_2)$, there are two cases.

If $x(p_{i+1}) \leq x(c_2)$, then since $y(p_i) \geq y(p_{i+1})$ and $x(p_i) < x(p_{i+1})$, it holds that $d_p(c_2, p_{i+1}) \leq d_p(c_2, p_i) \leq$

[1] The result is based on the assumption that the two sums of square roots of integers can be compared in polynomial time. The algorithm can be extended to L_1 and L_∞ cases without the assumption [14].

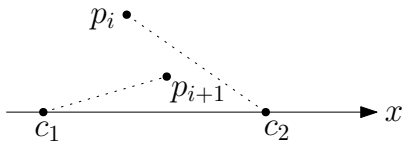

Figure 1: Illustrating the case where p_i is served by c_2 and p_{i+1} is served by c_1.

r_2. Hence, we can use c_2 to serve p_{i+1} without increasing the total cost of the solution.

If $x(p_{i+1}) > x(c_2)$, since $y(p_i) \geq y(p_{i+1})$ and $x(p_i) < x(p_{i+1})$, the two line segments $\overline{p_i c_2}$ and $\overline{p_{i+1} c_1}$ cross each other (e.g., see Fig. 2). By triangle inequality of the metric space, we have the following

$$d_p(c_1, p_i) + d_p(c_2, p_{i+1}) \leq d_p(c_2, p_i) + d_p(c_1, p_{i+1}). \quad (1)$$

If $d_p(c_2, p_{i+1}) \leq d_p(c_2, p_i)$, then since $d_p(c_2, p_i) \leq r_2$, we obtain $d_p(c_2, p_{i+1}) \leq r_2$. Thus, we can use c_2 to serve p_{i+1} without increasing the total cost of the solution. Otherwise, by Equation (1), we can derive $d_p(c_1, p_i) < d_p(c_1, p_{i+1})$, which implies that we can use c_1 to serve p_i without increasing the total cost of the solution.

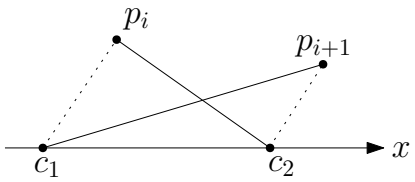

Figure 2: The two segments $\overline{p_i c_2}$ and $\overline{p_{i+1} c_1}$ cross each other. By triangle inequality, the sum of the lengths of the two solid segments is larger than or equal to the sum of the lengths of the two dotted segments.

The above shows that our switch operation "fixed" a conflict pair without increasing the total cost of the solution. If after the switch operation the new optimal solution still does not satisfy the lemma statement, then there must exist another conflict pair and we can continue applying the switch operation on them. Note that this procedure will be finite and thus eventually we will obtain an optimal solution without any conflict pairs, which implies that the optimal solution satisfies the lemma statement. □

Based on Lemma 1, we define a DAG G as follows. The vertex set consists of $n+1$ vertices $v_0, ...v_n$ so that each vertex v_i corresponds to an imaginary point between p_i and p_{i+1} (v_0 is to the left of p_1 and v_n is to the right of p_n). For all $0 \leq i < j \leq n$, we create a directed edge $e(i, j)$ from v_i to v_j, and the weight $w(i, j)$ of the edge is defined as $f(r)$, where r is the radius of the smallest disk centered at L that can cover all points of $P[i+1, j]$ (which is $\{p_{i+1}, p_{i+2}, \ldots, p_j\}$). Lemma 1 immediately leads to the following result.

Corollary 2 *A shortest path π from v_0 to v_n in G corresponds to an optimal solution to the disk coverage problem, i.e., the length of π is equal to the total cost and each edge $e(i, j)$ corresponds to a smallest disk centered at L covering all points of $P[i+1, j]$.*

Since G is a DAG and has $O(n^2)$ edges, a shortest path from v_0 to v_n can be computed in $O(n^2)$ time by a dynamic programming algorithm [12] if the weights of all graph edges are known. In the following, we show that the weights of all edges of G can be computed in $O(n^2)$ time. In particular, we have the following lemma.

Lemma 3 *For each vertex v_i, the weights of all its outgoing edges, i.e., $w(i, j)$ for all $j \in [i+1, n]$, can be computed in $O(n - i)$ time.*

Proof. To simplify the notation, we only consider the case $i = 0$. The algorithm can be generalized to any other index i in a straightforward manner. Our goal is to compute $w(0, j)$ for all $j \in [1, n]$ in $O(n)$ time.

For each $j \in [1, n]$, define c_j and r_j respectively as the center and the radius of the smallest disk centered at L that covers all points of $P[1, j]$. By definition, $w(0, j) = f(r_j)$. Below, we will give an incremental algorithm to compute c_j and r_j for all $j = 1, 2, \ldots, n$ in a total of $O(n)$ time.

Note that since $x(p_1) < x(p_2) < \cdots < x(p_n)$, we have $x(c_1) \leq x(c_2) \leq \cdots \leq x(c_n)$. This implies that when computing c_j from $j = 1$ to $j = n$, we only need to consider the locations of L from left to right.

For any two points p_i and p_j of P with $i < j$, there is a point, denoted by $q(i, j)$, on L such that for any point $c \in L$, $d_p(c, p_i) \leq d_p(c, p_j)$ if c is to the left of $q(i, j)$ and $d_p(c, p_i) \geq d_p(c, p_j)$ otherwise. We assume that given p_i and p_j, $q(i, j)$ can be computed in $O(1)$ time.

For any point p_i, we use p'_i to denote the point on L with the same x-coordinate as p_i. Clearly, for each $j \in [1, n]$, $r_j \geq d_p(p'_j, p_j)$.

As a warm-up and for better understanding the rationale of our algorithm, we first show how to process the first two points p_1 and p_2 (to compute c_j and r_j for $j = 1, 2$).

Initially, when $j = 1$, c_1 is p'_1 and $r_1 = d_p(c_1, p_1)$. Next, consider $j = 2$. We first compute the point $q(1, 2)$. The two points p'_1 (which is also c_1) and p'_2 divide L into three parts, and depending on which part contains $q(1, 2)$, there are three cases.

If $x(q(1, 2)) \leq x(c_1)$, then $d_p(c_1, p_2) \leq d_p(c_1, p_1) = r_1$. Thus, $c_2 = c_1$ and $r_2 = r_1$. Further, the point p_2 can be ignored in the future algorithm. Indeed, for any point $c \in L$ to the right of c_1, it holds that $d_p(c, p_1) \geq d_p(c, p_2)$. Since $x(c_j) \geq x(c_2)$ for all $j \geq 3$, when computing c_j for any $j \geq 3$, p_1 "dominates" p_2, and thus p_2 can be ignored.

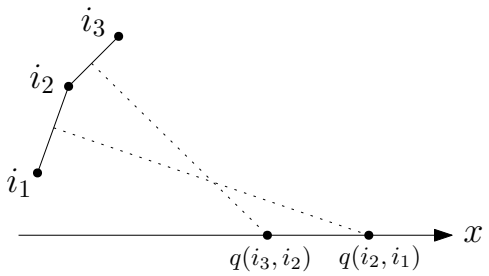

Figure 3: Illustrating a canonical list of three points in the L_2 metric. Each dotted segment is a perpendicular bisector of the segment connecting two consecutive points in the canonical list.

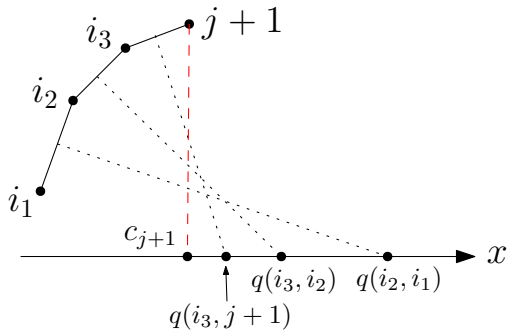

Figure 4: Update the canonical list of Fig. 3 by adding p_{j+1} to the end (i.e., setting $i_4 = j + 1$).

If $x(c_1) < x(q(1,2)) \leq x(p_2)$, then $c_2 = q(1,2)$ and $r_2 = d_p(c_2, p_1) = d_p(c_2, p_2)$. Further, as the above argument, p_2 can be ignored in the future algorithm.

If $x(q(1,2)) > x(p_2)$, then $d_p(p'_2, p_1) \leq d_p(p'_2, p_2)$. Thus, $c_2 = p'_2$ and $r_2 = d_p(p'_2, p_2)$. However, in this case, neither p_1 nor p_2 should be ignored because we do not know whether c_j is to the left or right of $q(1,2)$, e.g., for $j = 3$.

The above discusses our algorithm for processing p_1 and p_2. In the following, we describe our general algorithm. The pseudocode is given in Algorithm 1.

Suppose our algorithm has processed the points $p_1, p_2, \ldots, p_j$ (and thus c_i and r_i for all $i \in [1, j]$ have been computed) and is about to preprocess p_{j+1}. Then, our algorithm maintains a *canonical list* of h points $p_{i_1}, p_{i_2}, \ldots, p_{i_h}$ for $h \leq j$ with the following invariants (e.g., see Fig. 3).

1. $i_1 < i_2 < \cdots < i_h$.
2. $x(q(i_h, i_{h-1})) < x(q(i_{h-1}, i_{h-2})) < \cdots < x(q(i_2, i_1))$.
3. $x(p_{i_h}) \leq x(c_j) < x(q(i_h, i_{h-1}))$.
4. $r_j = d_p(c_j, p_{i_h})$.
5. To simplify the discussion, let $q(i_{h+1}, i_h) = c_j$ and $q(i_1, i_0)$ be the point on L with x-coordinate $+\infty$. For each $t \in [1, h]$, with respect to any point c between $q(i_{t+1}, i_t)$ and $q(i_t, i_{t-1})$ on L, the point p_{i_t} dominates all other points of $P[1, j]$, i.e., $d_p(c, p_{i_t}) \geq d_p(c, p_i)$ for all $i \in [1, j]$.

Initially, after p_1 is processed, our canonical list consists of a single point p_1, and all algorithm invariants hold (as an exercise, one can check that after p_2 is processed as discussed above the invariants also hold). Next, we discuss a general step of our algorithm for processing p_{j+1}.

We first compute the point $q(i_h, j+1)$ on L, and depending on its location with respect to c_j and $q(i_h, i_{h-1})$, there are three cases. Note that according to our algorithm invariants, $x(p_{i_h}) \leq x(c_j) < x(q(i_h, i_{h-1}))$.

If $x(q(i_h, j+1)) \leq x(c_j)$, then $d_p(c_j, p_{j+1}) \leq d_p(c_j, p_{i_h}) = r_j$. By definition, the disk centered at c_j with radius r_j is the smallest one covering all points of $P[1, j]$. As $d_p(c_j, p_{j+1}) \leq r_j$, the disk also covers p_{j+1} and thus is the smallest one covering all points of $P[1, j+1]$. Hence, $c_{j+1} = c_j$ and $r_{j+1} = r_j$. Further, for any point c on L to the right of c_{j+1}, $d_p(c, p_{i_h}) \geq d_p(c, p_{j+1})$, and thus p_{j+1} is dominated by p_{i_h} and can be ignored in the future algorithm. Therefore, in this case the canonical list $p_{i_1}, p_{i_2}, \ldots, p_{i_h}$ does not change, and all algorithm invariants hold since $c_{j+1} = c_j$.

If $x(c_j) < x(q(i_h, j+1)) < x(q(i_h, i_{h-1}))$ (this includes the case $h = 1$; recall that we have assumed $x(q(i_1, i_0)) = +\infty$), depending on whether $q(i_h, j+1)$ is to the left of p'_{j+1}, there are two subcases.

1. If $x(q(i_h, j+1)) \leq x(p_{j+1})$, then $c_{j+1} = q(i_h, j+1)$ and $r_{j+1} = d_p(c_{j+1}, p_{i_h})$. Indeed, by our algorithm invariants, since $x(c_j) < x(c_{j+1}) < x(q(i_h, i_{h-1}))$, c_{j+1} covers all points of $P[1, j]$ with distance r_{j+1}. On the other hand, by the definition of c_{j+1} and r_{j+1}, the disk centered at c_{j+1} with radius r_{j+1} is the smallest one covering p_{i_h} and p_{j+1}.

   Further, for any point c to the right of c_{j+1}, $d_p(c, p_{i_h}) \geq d_p(c, p_{j+1})$, and thus p_{j+1} is dominated by p_{i_h} and can be ignored in the future algorithm. Therefore, in this case the canonical list does not change, and all algorithm variants still hold since $x(c_j) < x(c_{j+1}) < x(q(i_h, i_{h-1}))$.

2. If $x(q(i_h, j+1)) > x(p_{j+1})$, then again by our algorithm invariants, $c_{j+1} = p'_{j+1}$ and $r_{j+1} = d_p(c_{j+1}, p_{j+1})$. In this case, we add p_{j+1} to the end of our canonical list by setting $i_{h+1} = j + 1$ and incrementing h by one (e.g., see Fig. 4). Due to $x(c_{j+1}) < x(q(i_h, j+1)) < x(q(i_h, i_{h-1}))$, one can verify that all our algorithm invariants hold.

If $x(q(i_h, j+1)) \geq x(q(i_h, i_{h-1}))$, then observe that p_{i_h} is dominated by p_{j+1} with respect to any location $c \in L$ to the left of $q(i_h, i_{h-1})$. Further, according to our algorithm invariants, p_{i_h} is dominated by at least one point of p_{i_t} for $t \in [1, h-1]$ with respect to any location $c \in L$ to the right of $q(i_h, i_{h-1})$. Hence, in this case we remove p_{i_h} from the canonical list, by decrementing h by one. In the following discussion, we assume h has been decremented and thus use $p_{i_{h+1}}$ to denote the removed point. We consider the location of $q(i_h, j+1)$ (again, this h has been decremented) and proceed as follows. As $x(q(i_{h+1}, j+1)) \geq x(q(i_{h+1}, i_h))$, $d_p(q(i_{h+1}, i_h), p_{j+1}) \geq d_p(q(i_{h+1}, i_h), p_{i_{h+1}}) = d_p(q(i_{h+1}, i_h), p_{i_h})$. Hence, $q(i_h, j+1)$ is to the right of $q(i_{h+1}, i_h)$. Since $x(q(i_{h+1}, i_h)) > x(c_j)$ (by algorithm invariants), we obtain $x(q(i_h, j+1)) > x(c_j)$. If $h > 1$ and $x(q(i_h, j+1)) \geq x(q(i_h, i_{h-1}))$, then we repeat the same procedure as above. Otherwise, depending on whether $x(q(i_h, j+1)) \leq x(p_{j+1})$, there are two subcases, whose processing is the same as above for the case $x(c_j) < x(q(i_h, j+1)) < x(q(i_h, i_{h-1}))$, and we omit the details.

The above describes a general step of our algorithm for processing p_{j+1}. The algorithm stops once p_n is processed. For the running time, processing p_{j+1} takes $O(1+t)$ time, where t is the number of points removed from the canonical list. Observe that each point of P will be added to the list and removed from the list at most once in the entire algorithm. Therefore, the total time of the algorithm is $O(n)$. □

Algorithm 1: Computing c_j and r_j for all $j \in [1, n]$

```
1  c_1 ← p'_1, r_1 ← d_p(p_1, p'_1), i_1 ← 1, h ← 1;
2  for j ← 1 to n − 1 do
3      compute q(i_h, j + 1);
4      if x(q(i_h, j + 1)) ≤ x(c_j) then
5          c_{j+1} ← c_j, r_{j+1} ← r_j;
6      else /* The following combines the
           second and third cases in the algorithm
           description                          */
7          while h > 1 and
             x(q(i_h, j + 1)) ≥ x(q(i_h, i_{h−1})) do
8              h ← h − 1, compute q(i_h, j + 1);
9          if x(q(i_h, j + 1)) ≤ x(p_{j+1}) then
10             c_{j+1} ← q(i_h, j + 1), r_{j+1} ← d_p(c_{j+1}, p_{i_h});
11         else
12             c_{j+1} ← p'_{j+1}, r_{j+1} ← d_p(c_{j+1}, p_{j+1}),
                 i_{h+1} ← j + 1, h ← h + 1;
```

By Lemma 3, we can compute a shortest path from v_0 to v_n in G in $O(n^2)$ time, after which an optimal solution for our original problem can be readily obtained according to Corollary 2. Note that the shortest path algorithm can be implemented in $O(n)$ space. Indeed, whenever a vertex v_i is processed, it is sufficient to know the weights of the outgoing edges of v_i by applying Lemma 3, and the weights of other edges of the graph can be ignored. Thus, we have the following theorem.

Theorem 4 *In any fixed L_p metric, for any $\alpha \geq 1$, the 1.5D disk coverage problem can be solved in $O(n^2)$ time and $O(n)$ space.*

4 The One-Dimensional Problem

In this section, we consider the two variations of the 1D problem. Note that in the 1D problem, for any two points q and q' on the x-axis, $d_p(q, q') = |x(q) - x(q')|$ in any L_p metric. Therefore, we will use $d(q, q')$ to denote the value $|x(q) - x(q')|$. We begin with the k-interval coverage problem.

4.1 The k-Interval Coverage Problem

In this problem, we are given a set P of n points on L (the x-axis), an integer $k \in [1, n]$, and $\alpha \geq 1$. The goal is to compute a set of at most k disks centered at L covering all points of P such that the total cost of all disks is minimized.

We follow the same notation as before and use $p_1, p_2, \ldots, p_n$ as the sorted list of P from left to right. Observe that Lemma 1 still holds for this problem. Thus, we build the same DAG G as before. The weights of the edges of G are also defined in the same way as before. Consequently, our problem is equivalent to computing a shortest path from v_0 to v_n in G with at most k edges (this is usually called a *k-link shortest path*). Further, we have a simple algorithm to compute the edge weights of the graph, as shown in the following lemma.

Lemma 5 *For any edge $e(i, j)$ with $0 \leq i < j \leq n$, the weight $w(i, j)$ can be computed in constant time.*

Proof. According to the definition, $w(i, j) = f(r) = r^\alpha$, where r is the radius of the smallest disk centered at L covering all points of $P[i+1, j]$. Observe that $r = |x(p_j) - x(p_{i+1})|/2$. Hence, $w(i, j)$ can be computed in constant time. □

Using Lemma 5, we can find a k-link shortest path from v_0 to v_n in G in $O(kn^2)$ time by a straightforward dynamic programming algorithm. However, we can do better due to that the edge weights of the graph obey the concave Monge condition [1, 2], which is proved in the following lemma.

Lemma 6 *The graph G has the concave Monge property, i.e., for any i and j with $0 < i+1 < j < n$, it holds that $w(i, j) + w(i+1, j+1) \leq w(i, j+1) + w(i+1, j)$.*

Proof. For any i', j' with $1 \leq i' \leq j' \leq n$, define $r(i', j') = |x(p_{j'}) - x(p_{i'})|/2$.

As discussed in the proof of Lemma 5, we have $w(i,j) = (r(i+1,j))^\alpha$, $w(i+1,j+1) = (r(i+2,j+1))^\alpha$, $w(i,j+1) = (r(i+1,j+1))^\alpha$, and $w(i+1,j) = (r(i+2,j))^\alpha$. Observe that $r(i+1,j)+r(i+2,j+1) = r(i+1,j+1)+r(i+2,j)$. Since $r(i+1,j+1) > r(i+1,j)$, $r(i+1,j+1) > r(i+2,j+1)$, and $\alpha \geq 1$, we can obtain that $(r(i+1,j))^\alpha + (r(i+2,j+1))^\alpha \leq (r(i+1,j+1))^\alpha + (r(i+2,j))^\alpha$. Therefore, $w(i,j)+w(i+1,j+1) \leq w(i,j+1) + w(i+1,j)$ holds. □

Due to the concave Monge property, we can resort to faster algorithms for computing a k-link shortest path from v_0 to v_n in G in $O(nk)$ [1], $O(n\sqrt{k \log n} + n \log n)$ [2], or $n2^{O(\sqrt{\log k \log\log n})}$ time [22]. Note that when applying these algorithms, we will not compute the graph G explicitly; rather, whenever the algorithm needs an edge weight, we use the algorithm in Lemma 5 to compute it in $O(1)$ time. Therefore, we have the following result.

Theorem 7 *For any $\alpha \geq 1$, after the points of P are sorted on L, the k-interval coverage problem can be solved in $\min\{O(nk), O(n\sqrt{k \log n} + n \log n), n2^{O(\sqrt{\log k \log\log n})}\}$ time.*

4.2 The Discrete Interval Coverage Problem

In this problem, we are given a set P of n points and another set Q of m points on L (the x-axis), as well as $\alpha \geq 1$. The goal is to compute a set of disks centered at the points of Q to cover all points of P such that the total cost of the disks is minimized.

Again, let $p_1, p_2, \ldots, p_n$ be the sorted list of P from left to right. We also sort all points of Q from left to right on L, and let $q_1, q_2, \ldots, q_m$ be the sorted list. One can verify that Lemma 1 still applies. With respect to the sorted list of P, we define the same DAG G as before. Here, the weight $w(i,j)$ of each edge $e(i,j)$ is defined as the smallest disk centered at a point in Q covering all points of $P[i+1, j]$. Hence, our problem is equivalent to finding a shortest path from v_0 to v_n in G. The following lemma gives an algorithm for computing the weights of the edges of G.

Lemma 8 *For each vertex v_i, the weights of all its outgoing edges, i.e., $w(i,j)$ for all $j \in [i+1, n]$, can be computed in $O(m+n-i)$ time.*

Proof. For any $j \in [i+1, n]$, let D be the smallest disk covering all points of $P[i+1, j]$ such that the center is a point in Q. Let $q(i+1, j)$ be the middle point of the line segment $\overline{p_{i+1}p_j}$ on L. Let c a point of Q that is closest to $q(i+1, j)$, and $r = \max\{|x(c) - x(p_{i+1})|, |x(c) - x(p_j)|\}$. Observe that c must be a center of D and r must be the radius. Therefore, to compute the weight $w(i,j)$, it is sufficient to determine the point of Q closest to $q(i+1, j)$.

We first compute the points $q(i+1, j)$ for all $j \in [i+1, n]$ in $O(n-i)$ time. Then the points of Q closest to $q(i+1, j)$'s for all $j \in [i+1, n]$ can be found in $O(m+n-i)$ time by a linear scan simultaneously on both the sorted list of Q and the list $q(i+1, i+1), q(i+1, i+2), \ldots, q(i+1, n)$, which is also sorted on L from left to right. Consequently, the weights $w(i,j)$ for all $j \in [i+1, n]$ can be computed in $O(m+n-i)$ time. □

By Lemma 8, we can compute a shortest path from v_0 to v_n in G in $O(n(m+n))$ time, after which an optimal solution for our original problem can be readily obtained. As in Section 3, with Lemma 8, the algorithm can be implemented in $O(n+m)$ space. Therefore, we have the following theorem, where the $O(m \log m)$ factor is due to the sorting of Q.

Theorem 9 *For any $\alpha \geq 1$, the discrete interval coverage problem can be solved in $O(n(m+n) + m \log m)$ time and $O(m+n)$ space.*

5 Concluding Remarks

In this paper, we present new algorithms for covering points by disks. We have been considering the cost function $f(r) = r^\alpha$ for a constant $\alpha \geq 1$. In fact, our algorithms for the 1.5D case and for the discrete 1D case also work with the same complexities for any non-decreasing function $f(r)$ as long as the following assumption holds: given any r, $f(r)$ can be computed in constant time. Our algorithm for the k-interval coverage problem, however, may not work for all non-decreasing functions, because the Monge property in Lemma 6 may not hold any more (in which case we can still use the straightforward $O(kn^2)$ time dynamic programming algorithm to solve the problem).

In addition, for the 1.5D case and the discrete 1D case, if there is an upper bound k on the number of disks that are allowed to be used, then the problem is equivalent to computing a k-link shortest path from v_0 to v_n in the DAG G, which can be done in $O(kn^2)$ time by dynamic programming after the graph G is computed.

It would be interesting to see whether the algorithms can be further improved, especially for the 1.5D problem and the discrete 1D problem. One might wonder whether the DAGs for these two problems also have Monge properties (either convex or concave). Unfortunately, we have found examples showing that the DAG for each problem does not have either convex or concave Monge property. Therefore, new techniques may be needed for further improvement.

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilbur. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[2] A. Aggarwal, B. Schieber, and T. Tokuyama. Finding a minimum weight k-link path in graphs with concave monge property and applications. *Discrete & Computational Geometry*, 12:263–280, 1994.

[3] H. Alt, E. Arkin, H. Brönnimann, J. Erickson, S. Fekete, C. Knauer, J. Lenchner, J. Mitchell, and K. Whittlesey. Minimum-cost coverage of point sets by disks. In *Proc. of the 22nd Annual Symposium on Computational Geometry*, pages 449–458, 2006.

[4] V. Auletta, D. Parente, and G. Persiano. Placing resources on a growing line. *Journal of Algorithms*, 26(1):87–100, 1998.

[5] B. Behsaz and M. Salavatipour. On minimum sum of radii and diameters clustering. *Algorithmica*, 73:143–165, 2015.

[6] V. Bilò, I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *Proc. of the 13th European Symposium on Algorithms*, pages 460–471, 2005.

[7] A. Biniaz, P. Bose, P. Carmi, A. Maheshwari, I. Munro, and M. Smid. Faster algorithms for some optimization problems on collinear points. 2018. To appear in *the 34th International Symposium on Computational Geometry*, full version at arXiv:1802.09505.

[8] P. Brass, C. Knauer, H.-S. Na, C.-S. Shin, and A. Vigneron. The aligned k-center problem. *International Journal of Computational Geometry and Applications*, 21:157–178, 2011.

[9] T. Chan. More planar two-center algorithms. *Computational Geometry: Theory and Applications*, 13(3):189–198, 1999.

[10] D. Chen, J. Li, and H. Wang. Efficient algorithms for the one-dimensional k-center problem. *Theoretical Computer Science*, 592:135–142, 2015.

[11] D. Chen and H. Wang. New algorithms for facility location problems on the real line. *Algorithmica*, 69:370–383, 2014.

[12] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3nd edition, 2009.

[13] H. Fan, M. Li, X. Sun, P. Wan, and Y. Zhao. Barrier coverage by sensors with adjustable ranges. *ACM Transactions on Sensor Networks*, 11:14:1–14:20, 2014.

[14] M. Gibson, G. Kanade, E. Krohn, I. Pirwani, and K. Varadarajan. On clustering to minimize the sum of radii. *SIAM Journal on Computing*, 41:47–60, 2012.

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

[16] A. Karmakar, S. Das, S. Nandy, and B. Bhattacharya. Some variations on constrained minimum enclosing circle problem. *Journal of Combinatorial Optimization*, 25(2):176–190, 2013.

[17] N. Lev-Tov and D. Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47:489–501, 2005.

[18] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[19] N. Megiddo and K. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13:182–196, 1984.

[20] N. Megiddo and A. Tamir. New results on the complexity of p-centre problems. *SIAM Journal on Computing*, 12(4):751–758, 1983.

[21] K. Pahlavan and A. Levesque. *Wireless Information Networks*. Wiley, New York, NY, 2nd edition, 2005.

[22] B. Schieber. Computing a minimum weight k-link path in graphs with the concave monge property. *Journal of Algorithms*, 29(2):204–222, 1998.

[23] H. Wang and J. Zhang. Line-constrained k-median, k-means, and k-center problems in the plane. *International Journal of Computational Geometry and Applications,*, 26:185–210, 2016.