The LR-dispersion problem

Toshihiro Akagii*  Tetsuya Araki†  Shin-ichi Nakano*

1 Introduction

The facility location problem and many of its variants have been studied[7, 8]. A typical problem is to find a set of locations to place facilities with the designated cost minimized. By contrast, in this paper we consider the dispersion problem, which finds a set of locations with the designed cost maximized.

Given a set \( P \) of \( n \) points, and the distance \( d \) for each pair of points, and an integer \( k \) with \( k \leq n \), we wish to find a subset \( S \subset P \) with \( |S| = k \) such that some designated cost is maximized[1, 5, 6, 10, 11, 12, 13].

In one of typical cases the cost to be maximized is the minimum distance between two points in \( S \). If \( P \) is a set of points on the plane then the problem is NP-hard[12, 13], and if \( P \) is a set of points on the line then the problem can be solved in \( O(max\{n \log n, kn\}) \) time[12, 13] by dynamic programming approach, and in \( O(n \log n) \) time[1] by sorted matrix search method[4, 9].

In this paper we consider two variants of the dispersion problem on the line. Let \( P \) be a set of points on the horizontal line. We wish to find a subset \( S \subset P \) with \( |S| = k \) maximizing cost(\( S \)) defined as follows.

Let the cost cost(\( s \)) \( s \in S = \{s_1, s_2, \ldots, s_k\} \) be the sum of the distance to its left neighbor in \( S \) and the distance to its right neighbor in \( S \). We assume \( s_1, s_2, \ldots, s_k \) are sorted from left to right. Especially the leftmost point \( s_1 \in S \) has no left neighbor, so we define the cost of \( s_1 \) is \( d(s_1, s_2) \). Similarly the cost of the rightmost point \( s_k \) is \( d(s_k, s_{k-1}) \). And the cost(\( S \)) of \( S \) is the minimum cost among the costs cost(\( s_1 \)), cost(\( s_2 \)), \ldots, cost(\( s_k \)). We call the problem above the LR-dispersion problem. An \( O(kn^2 \log n) \) time algorithm based on dynamic programming is known[2].

In this paper we design an algorithm to solve the LR-dispersion problem. Our algorithm runs in \( O(n \log n) \) time, and based on matrix search method[4, 9].

2 \((\lambda, k)\)-LR-dispersion

In this section we give a linear time algorithm to solve a decision version of the LR-dispersion problem.

Given a set \( P = \{p_1, p_2, \ldots, p_n\} \) of points on a horizontal line, and two numbers \( k \) and \( \lambda \) we wish to decide if there exists a subset \( S \subset P \) with \( |S| = k \) and cost(\( S \)) \( \geq \lambda \). We call the problem as the \((\lambda, k)\)-LR-dispersion problem. We have the following lemma.

**Lemma 1.** If \((\lambda, k)\)-LR-dispersion problem has a solution \( S = \{s_1, s_2, \ldots, s_k\} \subset P \), then \( S' = \{p_1, s_2, s_3, \ldots, s_{k-1}, p_n\} \) is also a solution of the \((\lambda, k)\)-LR-dispersion problem.

**Proof.** Since cost(\( S \)) \( \leq \) cost(\( S' \)), if \( S \) is a solution then \( S' \) is also a solution and cost(\( S \)) = cost(\( S' \)).

The algorithm below is a greedy algorithm to solve the \((\lambda, k)\)-LR-dispersion problem. Note that cost(\( s_i \)) for \( i = 3, 4, \ldots, k-1 \) is \( d(s_{i-2}, s_i) \). By setting a dummy point \( s_0 = s_1 \), cost(\( s_2 \)) is also \( d(s_2, s_2) \). Also note that cost(\( k \)) = \( d(s_{k-1}, s_k) \).

**Algorithm 1** find \((\lambda, k)\)-LR-dispersion \((P, k, \lambda)\)

\[
\begin{align*}
&\text{/} * \ P = \{p_1, p_2, \ldots, p_n\} \text{ and } p_1, p_2, \ldots, p_n \text{ are sorted from left to right } * /
&\text{/} * \text{ Choose } s_1 \text{ and } s_k * /
& s_1 = p_1, s_k = p_n
& s_0 = s_1 \quad \text{/} * \text{ Dummy } * /
&\text{/} * \text{ Choose } s_2, s_3, \ldots, s_{k-1} * /
& c = 2
&\text{for } i = 2 \text{ to } k - 1 \text{ do}
& \quad \text{while } d(s_{i-2}, p_c) < \lambda \text{ and } d(p_c, p_n) \geq \lambda \text{ do}
& \quad \quad c++
& \quad \text{end while}
& \quad \text{if } d(p_c, p_n) < \lambda \text{ then}
& \quad \quad \text{/} * \text{ no solution since } d(p_c, p_n) < \lambda * /
& \quad \quad \text{return NO}
& \quad \text{else}
& \quad \quad \text{/} * \ d(s_{i-2}, p_c) \geq \lambda \text{ holds } * /
& \quad \quad s_i = p_c \quad \quad \text{/} * s_i \text{ is found } * /
& \quad \quad c++
& \quad \text{end if}
&\text{end for}
&\text{/} * \text{ Output } * /
&\text{return } S = \{s_1, s_2, \ldots, s_k\}
\end{align*}
\]

Now we prove the correctness of the algorithm. Assume for a contradiction that the algorithm output NO for a given problem but it has a solution.

Let \( G = \{g_1, g_2, \ldots, g_k\} \) with \( k' < k \) be the points chosen by the algorithm, and \( O = \{o_1, o_2, \ldots, o_k\} \) the

*Department of Computer Science, Gunma University
†National Institute of Informatics, Japan
points of a solution. By Lemma 1 we can assume $o_1 = p_1$ and $o_k = p_n$. Note that $g_1 = o_1 = p_1$ and $g_k = o_k = p_n$ hold. We have the following two cases.

**Case 1**: For all $i$, $1 \leq i < k'$, $g_i \leq o_i$ holds.

Then our greedy algorithm can choose at least one more point $o_{k'}$ or more left point. A contradiction.

**Case 2**: For some $i$, $1 \leq i < k'$, $g_i > o_i$ holds.

Since $g_2$ is chosen in a greedy manner, we can assume $g_2 \leq o_2$. Let $j$ be the minimum such $i$. Since $g_1 - 2 \leq o_j - 2$ and $g_1 - 1 \leq o_1 - 1$ hold, our greedy algorithm choose $o_i$ or more left point as $g_i$. A contradiction.

**Theorem 1.** One can solve the decision version of the LR-dispersion problem in $O(n)$ time.

### 3 LR-dispersion

One can design an $O(n \log n)$ time algorithm to solve the LR-dispersion problem, based on the sorted matrix search method[4, 9]. See the long version[3] for detail.

**Theorem 2.** One can solve the LR-dispersion problem in $O(n \log n)$ time.

### 4 Generalization

In this section we consider one more variant of the dispersion problem and give an algorithm to solve the problem, which runs in $O(n \log n)$ time. In the original dispersion problem the cost is the minimum distance between two points $s_i$ and $s_{i+1}$. We generalize this to the minimum distance between $s_i$ and $s_{i+h}$, for given $h$.

Given a set $P = \{p_1, p_2, \ldots, p_n\}$ of points on a horizontal line, and a distance $d$ for each pair of points, and two integers $k$, and $h$ with $k, h \leq n$, we wish to find a subset $S = \{s_1, s_2, \ldots, s_k\} \subset P$ maximizing $\text{cost}(S)$ defined as follows.

$$L\text{cost}(S) = \min \{d(s_1, s_2), d(s_1, s_2), \ldots, d(s_1, s_k)\},$$

$$R\text{cost}(S) = \min \{d(s_{k-h+1}, s_k), d(s_{k-h+2}, s_k), \ldots, d(s_{k-h}, s_k)\},$$

$$M\text{cost}(S) = \min \{d(s_1, s_{1+h}), d(s_2, s_{2+h}), \ldots, d(s_{k-h}, s_k)\},$$

then $\text{cost}(S) = \min \{L\text{cost}(S), R\text{cost}(S), M\text{cost}(S)\}$.

We call the problem above the $h$-dispersion problem. The original dispersion problem on the line is the $h$-dispersion problem with $h = 1$ and the LR-dispersion problem is the $h$-dispersion problem with $h = 2$.

See the long version[3] for detail.

**Theorem 3.** One can solve the $h$-dispersion problem in $O(n \log n)$ time.

### 5 Conclusion

In this paper we have presented two algorithms to solve the LR-dispersion problem and the $h$-dispersion problem. The running time of the algorithms are $O(n \log n)$.

An $O(n \log \log n)$ time algorithm to solve the original dispersion problem on the line is known[1]. Can we design an $O(n \log \log n)$ time algorithm to solve the $h$-dispersion problem?

### References


