数理解析研究所講究録
第2040巻 2017年 27-34

The LR-dispersion problem

Toshihiro Akagi
Department of Computer Science, Gunma University
Tetsuya Araki
Principles of Informatics Research Division,
National Institute of Informatics
Shin-ichi Nakano
Department of Computer Science, Gunma University

1 Introduction

The facility location problem and many of its variants have been studied[6, 7]. 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, 4, 5, 9, 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[11, 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[11, 13] by dynamic programming approach, and in $O(n \log\log n)$ time[1] by sorted matrix search method[3, 8].

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 $cost(s)$ of $s \in S = \{s_1, s_2, \ldots, s_k\}$ 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 $d(s_{k-1}, s_k)$. 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[3, 8].

The remainder of this paper is organized as follows. Section 2 contains an algorithm for the decision version of the LR-dispersion problem. Section 3 gives our algorithm for the LR-dispersion problem. Section 4 treats one more variant of the dispersion problem. Finally Section 5 is a conclusion.

2 (λ, 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 λ 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 (λ, k)-LR-dispersion problem.

We have the following lemma.

Lemma 1. If (λ, 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 (λ, k)-LR-dispersion problem.

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

The algorithm below is a greedy algorithm to solve the (λ, 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 $d(s_{2-2}, s_2) = d(s_1, s_2)$. Also note that $cost(k) = d(s_{k-1}, s_k)$.

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 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 $g_{j-2} \leq o_{j-2}$ and $g_{j-1} \leq o_{j-1}$ 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.

Algorithm 1 find (λ, k)-LR-dispersion (P, k, λ)

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

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[3, 8].

First let M be the matrix in which each element $m_{i,j}$ is $d(p_i, p_j)$ if $i < j$, otherwise 0. Then $m_{i,j} \leq m_{i,j+1}$ and $m_{i,j} \geq m_{i+1,j}$ always holds, so the elements in the rows and columns are sorted, respectively. The cost $cost(S)$ of a solution S of the LR-dispersion problem is some element in the matrix. We are going to find the largest λ in M for which the (λ, k)-LR-dispersion problem has a solution.

By appending a suitable number of large enough elements to M as the elements in the topmost rows and the rightmost columns we can assume n is a power of 2. Note that the elements in the rows and columns are still sorted, respectively. Let M be the resulting matrix. Our algorithm consists of rounds $s = 1, 2, \ldots, \log n$, and maintains a set L_s of (non-overlapping) submatrices of M possibly containing the optimal value λ^*. Hypothetically first we set $L_0 = \{M\}$. Assume we are now staring round s.

For each subset M in L_{s-1} we divide M into the four submatrices with $n/2^s$ rows and $n/2^s$ columns and put them into L_s. We never copy these submatrices. We just update the index of the corner elements of each submatrix.

Let λ_{min} be the median of the lower left corner elements of the submatrices in L_s. Then for the $\lambda = \lambda_{min}$ we solve the (λ, k)-LR-dispersion problem, using the algorithm in Section2. We have the following two cases.

If the (λ, k)-LR-dispersion problem has no solution then we remove from L_s each submatrix with the lower left corner element (the smallest element) greater than λ_{min}. Since $\lambda_{min} > \lambda^*$ holds, each removed submatrix has no chance to contain λ^*. Thus we can remove at least $|L_s|/2$ submatrices from L_s.

Otherwise if the (λ, k)-LR-dispersion problem has a solution then we remove from L_s each submatrix with the upper right corner element (the largest element) smaller than λ_{min}. Since $\lambda_{min} \leq \lambda^*$ holds, each removed submatrix has no chance to contain λ^*. Also if L_s has several submatrices with the upper right corner element equal to λ_{min} then we remove them except one from L_s. Now we can observe that, for each "chain" of submatrices, which is the sequence of submatrices in L_s with lower left to upper right diagonals on the same line, the number of submatrices (1) having the lower left corner element smaller than λ_{min} (2) but remaining in L_s, is at most one (since the elements on "the common diagonal line" are sorted). Thus, if $|L_s|/2 > D_s$, where $D_s = 2^{s+1}$ is the number of chains plus one, then we can remove at least $|L_s|/2 - D_s + 1$ submatrices from L_s.

Similarly let λ_{max} be the median of the upper right corner elements of submatrices in L_s, and for the $\lambda = \lambda_{max}$ we solve the (λ, k)-LR-dispersion problem and similarly remove some submatrices from L_s. This ends round s.

Now after round $\log n$ each matrix in $L_{\log n}$ has just one element, then we can find the λ^* using a binary search with the linear time decision algorithm in Section 2.

We can prove that at the end of round s the number of submatrices in L_s is at most $2D_s$, as follows.

First L_0 has 1 submatrix, which is less than $2D_0 = 4$. By induction assume that L_{s-1} has $2D_{s-1} = 2 \cdot 2^s$ submatrices.

At round s we first partite each submatrix in L_{s-1} into four submatrices then put them into L_s. Now the number of submatrices in L_s is at most $4 \cdot 2D_{s-1} = 4D_s$. We have four cases.

If the (λ, k)-LR-dispersion problem has no solution for $\lambda = \lambda_{min}$ then we can remove at least a half of the submatrices in L_s is at most $2D_s$, as desired. If the (λ, k)-LR-dispersion problem has a solution for $\lambda = \lambda_{max}$ then we can remove at least a half of the submatrices in L_s is at most $2D_s$, as desired. Otherwise if $|L_s|/2 \leq D_s$ then the number of the submatrices in L_s (even before the removal) is at most $2D_s$, as desired. Otherwise (1) after the check for $\lambda = \lambda_{min}$ we can remove at least $|L_s|/2 - D_s$ submatrices (consisting of too small elements) from L_s, and (2) after check for $\lambda = \lambda_{max}$ we can remove at least $|L_s|/2 - D_s$ submatrices (consisting of too large elements) from L_s, so the number of the remaining submatrices in L_s is at most $|L_s| - 2(|L_s|/2 - D_s) = 2D_s$, as desired.

Thus at the end of round s the number of submatrices in L_s is always at most $2D_s$, and at the end of round $\log n$ the number of submatrices is at most $2D_{\log n} = 4n$.

Now we consider the running time. We implicitly treat each submatrix as the index of the upper right element in M and the number of lows (= the number of columns). Except for the calls of the linear time decision algorithm for the (λ, k)-LR-dispersion problem, we need $O(|L_{s-1}|) = O(D_{s-1})$ time for each round $s = 1, 2, \ldots, \log n$, and $D_0 + D_1 + \cdots + D_{\log n - 1} = 2 + 4 + \cdots + 2^{\log n} < 2 \cdot 2^{\log n} = 2n$ holds, so this part needs $O(n)$ time in total. (Here we use the linear time algorithm to find the median.)

Since each round calls the linear time decision algorithm twice and the number of round is $\log n$ this part needs $O(n \log n)$ time in total.

After round $s = \log n$ each matrix has just one element. Then we can find the λ^* among the $|L_{\log n}| \leq 2D_{\log n} = 4n$ elements by (1) sorting them, then (2) performing binary search with the linear time decision algorithm at most $\log 4n$ times. This part needs $O(n \log n)$ time.

Thus the total running time is $O(n \log n)$. With a similar method we have solved the (original) dispersion problem in $O(n \log n)$ time[1].

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

$Lcost(S) = \min\{d(s_1, s_2), d(s_1, s_3), \ldots, d(s_1, s_h)\}$, $Rcost(S) = \min\{d(s_{k-h+1}, s_k), d(s_{k-h+2}, s_k), \ldots, d(s_{k-1}, s_k)\}$ and $Mcost(S) = \min\{d(s_1, s_{1+h}), d(s_2, s_{2+h}), \ldots, d(s_{k-h}, s_k)\}$ then $cost(S) = \min\{Lcost(S), Rcost(S), Mcost(S)\}$.

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 $h = 2$.

Lemma 2. If (λ, k)-h-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 (λ, k)-h-dispersion problem.

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

The algorithm below is a greedy algorithm to solve the problem. 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 points of a solution. By Lemma 2 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 points $o_{k'}$ or more left point. A contradiction.

Case 2 : For some i, $1 \leq i < k'$, $g_i > o_i$ holds. Since $g_2, g_3, \ldots, g_h$ are chosen in a greedy manner, we can assume $g_j \leq o_j$ for $j = 2, 3, \ldots, h$. Let j be the minimum such i. Since $g_{j-h} \leq o_{j-h}$, $g_{j-h+1} \leq o_{j-h+1}$, $\ldots$, $g_{j-1} \leq o_{j-1}$ hold, our greedy algorithm choose o_i or more left point as g_i. A contradiction.

Theorem 3. One can solve the decision version of the h-dispersion problem in $O(n)$ time.

Therefore, similar to the algorithm in Section 3, we can design $O(n \log n)$ time algorithm to solve the h-dispersion problem.

Theorem 4. 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 ?

Algorithm 2 find (λ, k)-h-dispersion (P, h, k, λ)

```
/* Choose $s_1$ and $s_k$ */
$s_1 = p_1$, $s_k = p_n$
/* Dummy */
$s_0 = s_1, s_{-1} = s_1, s_{-2} = s_1, \dots, s_{-h+2} = s_1$
/* Choose $s_2, s_3, \dots, s_{k-1}$ */
$c = 2$
for $i = 2$ to $k - 1$ do
  while $d(s_{i-h}, p_c) < \lambda$ and $d(p_c, p_n) \geq \lambda$ do
    $c++$
  end while
  if $d(p_c, p_n) < \lambda$ then
    /* no solution since $d(p_c, p_n) < \lambda$ */
    return NO
  else
    /* $d(s_{i-h}, p_c) \geq \lambda$ holds*/
    $s_i = p_c$                                        /* $s_i$ is found */
    $c++$
  end if
end for
/* Output */
return $S = \{s_1, s_2, \dots, s_k\}$
```

References

[1] T. Akagi and S. Nakano, Dispersion problem on the Line, Technical Report, 2016-AL-158-4, IPSJ (2016).

[2] T. Akagi, T. Araki and S. Nakano, Variants of the Dispersion Problem, Technical Report, 2017-AL-161-3, IPSJ (2017).

[3] P. Agarwal and M. Sharir, Efficient Algorithms for Geometric Optimization, Computing Surveys, 30, pp.412-458 (1998).

[4] C. Baur and S. P. Feketee, Approximation of Geometric Dispersion Problems, Pro. of APPROX '98, Page 63-75 (1998).

[5] B. Chandra and M. M. Halldorsson, Approximation Algorithms for Dispersion Problems, J. of Algorithms, 38, pp.438-465 (2001).

[6] Z. Drezner, Facility Location: A Survey of Applications and Methods, Springer (1995).

[7] Z. Drezner and H. W. Hamacher, Facility Location: Applications and Theory, Springer (2004).

[8] G. Frederickson, Optimal Algorithms for Tree Partitioning, Proc. of SODA'91 Pages 168-177 (1991).

[9] R. Hassin, S. Rubinstein and A. Tamir, Approximation Algorithms for Maximum Dispersion, Operation Research Letters, 21, pp.133-137 (1997).

[10] T. L. Lei and R. L. Church, On the unified dispersion problem: Efficient formulations and exact algorithms, European Journal of Operational Research, 241, pp.622-630 (2015).

[11] S. S. Ravi, D. J. Rosenkrantz and G. K. Tayi, Heuristic and Special Case Algorithms for Dispersion Problems, Operations Research, 42, pp.299-310 (1994).

[12] M. Sydow, Approximation Gurantees for Max Sum and Max Min Facility Dispersion with Parameterised Triangle Inequality and Applications in Result Diversification, Mathematica Applicanda, 42, pp.241-257 (2014).

[13] D. W. Wang and Y. S. Kou, A study on Two Geometric Location Problems, Information Processing Letters, 28, pp.281-286 (1988).