Vol.2021-AL-183 No.2
2021/5/7

k-Dispersion on Intervals

Tetsuya Araki[1,a)] Hiroyuki Miyata[1,b)] Shin-ichi Nakano[1,c)]

Abstract:
Given a set of n disjoint intervals on a line, and an integer k, we want to find k points in the intervals so that the minimum pairwise distance of the k points is maximized. Intuitively, given a set of n disjoint time intervals on a timeline, each of which is a time span we are allowed to check something, and an integer k, which is the number of times we will check something, we plan the k checking times so that the checks occur at equal time intervals as much as possible, that is, we want to maximize the minimum time interval between the k checking times. The problem is called the k-dispersion problem on intervals. If we need to choose exactly one point in each interval, so $k = n$, and the disjoint intervals are given in the sorted order on the line, then two $O(n)$ time algorithms to solve the problem are known.
In this paper we give the first $O(n)$ time algorithm to solve the problem for any constant k. Here one can check twice or more in one time interval. Our algorithm works even if the disjoint intervals are given in any (not sorted) order. If the disjoint intervals are given in the sorted order on the line, then, by slightly modifying the algorithm, one can solve the problem in $O(\log n)$ time. This is the first sublinear time algorithm to solve the problem. Also we show some results on the k-dispersion problem on disks, including a PTAS.
keywords: dispersion problem, algorithm

1. Introduction

The facility location problem and many of its variants have been studied [11], [12]. Typically, given a set of locations on which facilities can be placed and an integer k, we want to place k facilities on some locations so that a designated objective function is minimized. By contrast in the *dispersion problem*, we want to place facilities so that a designated objective function is maximized.

In this paper we consider the dispersion problem on intervals. Given a set of n disjoint intervals on a line, and an integer k, we want to find k points in the intervals so that the minimum pairwise distance of the k points is maximized. See an example in Fig. 1.

Intuitively, given a set of n disjoint (non-overlapping) time intervals on a timeline, each of which is a time span we are allowed to check something, and an integer k, which is the number of times we will check something, we plan the k checking times so that the checks occur at equal time intervals as much as possible, that is, we want to maximize the minimum time interval between the k checking times. We call the problem *the k-dispersion problem on intervals*. Let S be a set of optimal k points on the line (corresponding to the k checking times), and $cost(S) = \min_{\{s,t\}\subset S}\{d(s,t)\}$ the minimum pairwise distance of the k points in S.

Fig. 1 An example of the dispersion problem on intervals with $k = 6$.

1 Gunma University
Kiryu, 376–8515, Japan
a) tetsuya.araki@gunma-u.ac.jp
b) hmiyata@gunma-u.ac.jp
c) nakano@cs.gunma-u.ac.jp

If we need to choose exactly one point in each time interval, and so $k = n$, and the disjoint intervals are given in the sorted order on the line, two $O(n)$ time algorithms to solve the problem are known [5], [18].

Our result In this paper we give the first $O(n)$ time algorithm to solve the problem for any constant k. Here one can choose two or more points in one interval. Our algorithm works even if the disjoint intervals are given in any (unsorted) order. Our algorithms is based on the pigeonhole principle, and is a generalization of the algorithm in [3] to solve a similar dispersion problem.

If the disjoint intervals are given in the sorted order on the line, then, by slightly modifying the algorithm, one can solve the problem in $O(\log n)$ time. This is the first sublinear time algorithm to solve the problem.

Related result Given a set P of n possible locations, and a distance function d for each pair of locations, and an integer k with $k \ll n$, the *max-min k-dispersion problem* computes a subset $S \subset P$ with $|S| = k$ such that the cost $cost(S) = \min_{\{u,v\}\subset S}\{d(u,v)\}$ is maximized. Several results are known for this max-min k-dispersion problem [1], [2], [14], [19], [21], For the max-sum version several results are also known [4], [6], [8], [9], [10], [15], [17], [19]. For a variety of related problems, see [4], [10]. See more applications, including *result diversification*, in [9], [19], [20].

Given a set of disks, we want to choose one point in each disk so that the minimum distance among the points is maximized. The problem is called the *dispersion problem on disks*, and some results are known [7], [13], [16]. The k-dispersion problem on intervals is the 1D version of the dispersion problem on disks.

The remainder of this paper is organized as follows. In Section 2 we design an $O(n)$ time simple algorithm to solve the dispersion problem when intervals are given "unsorted" on a line. Section 3 gives an $O(\log n)$ time algorithm to solve the dispersion problem when intervals are given sorted on a line. In Section 4 we show some results on the k-dispersion problem on disks. Finally Section 5 is a conclusion.

2. k-dispersion for unsorted intervals

In this section we design a simple $O(n)$ time algorithm to solve the k-dispersion problem on intervals when the disjoint n intervals are given unsorted on a line. The idea of our algorithm is a simple divide and conquer algorithm using the pigeonhole principle, as follows. Similar idea is used to solve a similar max-min dispersion problem on a line [3].

Let I be a set of disjoint intervals on a horizontal line and p_ℓ and p_r are the leftmost point and the rightmost point in I. One can find p_ℓ and p_r in $O(n)$ time.

If $k = 1$ then a solution S of the 1-dispersion problem is $\{p_\ell\}$.

If $k = 2$ then the solution S of the 2-dispersion problem is $\{p_\ell, p_r\}$.

If $k = 3$ then let the solution S be $\{p_\ell, p_s, p_r\}$. The solution S consists of p_ℓ and p_r and exactly one more point p_s in some interval in I. We can compute p_s as follows.

Let $i_0 = p_\ell$, $i_2 = p_r$, and let i_1 be the midpoint between p_ℓ and p_r. If some interval in I contains i_1 then $p_s = i_1$. Otherwise, let U_1 be the interval (i_0, i_1), and U_2 be the interval (i_1, i_2). Now p_s appears in either U_1 or U_2. So, by pigeonhole principle, p_s does not appear in U_1 or U_2. Thus we have two cases.

Case 1: p_s does not appear in U_1.

In this case, S consists of p_ℓ and the solution of the 2-dispersion problem on intervals in $(i_1, i_2]$, say $R \subset I$, which consists of (1) the leftmost point in R and (2) p_r.

Case 2: p_s does not appear in U_2.

In this case, S consists of p_r and the solution of the 2-dispersion problem on intervals in $[i_0, i_1)$, say $L \subset I$, which consists of (1) the rightmost point in L and (2) p_ℓ.

We can generalize this method for a constant $k > 3$, as follows.

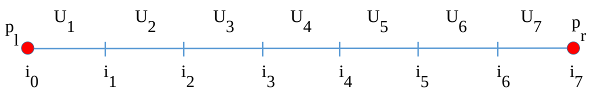

Fig. 2 Illustration of $U_1, U_2, \cdots$ for $k = 8$.

Algorithm 1 Find-dispersion-on-Intervals(I, k)

/* p_ℓ and p_r are the leftmost point and the rightmost point in I */
if $k = 1$ **then**
  $S = \{p_\ell\}$
  return S
end if
if $k = 2$ **then**
  $S = \{p_\ell, p_r\}$
  return S
end if
/* $i_0 = p_\ell$, $i_{k-1} = p_r$ and let $i_1, i_2, \cdots, i_{k-2}$ be the points which evenly spaced on the line between p_ℓ and p_r */
/* $k \geq 3$ */
if each of $i_1, i_2, \cdots, i_{k-2}$ is contained in some interval in I **then**
  $S = \{i_0, i_1, \cdots, i_{k-1}\}$
  return S
end if
/* Case: S has no point in $U_1 = (i_0, i_1])$ */
Let R be the set of intervals in $(i_1, i_{k-1}]$.
(if there is an interval in I containing i_1 then replace its left end to i_1)
$S_L = \{p_\ell\}$
S_R =**Find-dispersion-on-a-line($R, k-1$)**
$S = S_L \cup S_R$
/* Case: S has no point in $U_j = (i_{j-1}, i_j]$ for $j = 2, 3, \cdots, k-2$ */
for $j = 2$ to $k-2$ **do**
  Let L be the intervals in $[i_0, i_{j-1}]$.
  (If there is an interval in I containing i_{j-1} then replace its right end to i_{j-1})
  Let R be the intervals in $(i_j, i_{k-1}]$.
  (If there is an interval in I containing i_j then replace its left end to i_j)
  for $s = 1$ to $k-1$ **do**
    S_L =**Find-dispersion-on-a-line(L, s)**
    S_R =**Find-dispersion-on-a-line($R, k-s$)**
    if $cost(S_L \cup S_R) > cost(S)$ **then**
      $S = S_L \cup S_R$
    end if
  end for
end for
/* Case: S has no point in $U_{k-1} = (i_{k-2}, i_{k-1})$ */
Let L be the set of intervals in $[i_0, i_{k-2}]$.
(If there is an interval in I containing i_{k-2} then replace its right end to i_{k-2})
S_L =**Find-dispersion-on-a-line($L, k-1$)**
$S_R = \{p_r\}$
if $cost(S_L \cup S_R) > cost(S)$ **then**
  $S = S_L \cup S_R$
end if
return S

Let $i_0 = p_\ell$, $i_{k-1} = p_r$ and let $i_1, i_2, \cdots, i_{k-2}$ be the points which evenly spaced on the line between p_ℓ and p_r. Clearly the cost of the solution is at most $d(i_0, i_1)$, where $d(i_0, i_1)$ is the distance between i_0 and i_1, If each of $i_1, i_2, \cdots, i_{k-2}$ is contained in some interval in I, then $\{i_0, i_1, \cdots, i_{k-1}\}$ is the solution, and the cost is $d(i_0, i_1)$. Assume otherwise. Let U_j be the interval $(i_{j-1}, i_j]$ for $j = 1, 2, \cdots, k-2$, and U_{k-1} be the interval (i_{k-2}, i_{k-1}). See an example in Fig. 2.

The solution for the k-dispersion problem consists of p_ℓ and p_r and exactly $k-2$ points in (i_0, i_{k-1}). So, by pigeonhole principle, S has no point in at least one of U_1, U_2, $\cdots$, U_{k-1}. Thus we have $k-1$ cases as follows.

Case 1: S has no point in U_1.

If there is an interval in I containing i_1, then replace its left end to i_1.

In this case, S consists of (1) p_ℓ and (2) the solution of $(k-1)$-dispersion problem for the intervals in $[i_1, i_{k-1}]$.

Case 2: S has no point in U_2.

In this case, for some s with $1 \le s \le k-1$, S consists of (1) the solution of s-dispersion problem for the intervals, say L, in $[i_0, i_1]$ (if there is an interval in I containing i_1 then replace its right end to i_1) and (2) the solution of $(k-s)$-dispersion problem for the intervals, say R, in $[i_2, i_{k-1}]$ (if there is an interval in I containing i_2 then replace its left end to i_2). Note that since S has no point in U_2 the solution for L does not affect the solution for R, so we can solve the two smaller subproblems independently. Also note that if there is an interval in I containing both i_1 and i_2, then two subintervals of the interval appear one in L and the other in R.

Case 3: S has no point in U_3.

Similar to Case 2.

$\cdots$

Case $k-2$: S has no point in U_{k-2}.

Similar to Case 2.

Case $k-1$: S has no point in U_{k-1}.

If there is an interval in I containing i_{k-2}, then replace its right end to i_{k-2}.

In this case, S consists of (1) the solution of $(k-1)$-dispersion problem for the intervals in $[i_0, i_{k-2}]$ and (2) p_r.

We (recursively) check all possible cases and choose the best one. See algorithm **Find-dispersion-on-intervals**.

Thus if we have the solution of at most $2k^2$ smaller child dispersion problems then we can solve the original k-dispersion problem.

We have the following theorem.

Theorem 1. *One can solve the k-dispersion problem on intervals in $O(n)$ time even when the intervals are given unsorted on a line.*

Proof. Consider the tree structure of the recursive calls. Each inner node has at most $2k^2$ children and the height of the tree is at most k, so the number of inner node is at most $(2k^2)^k$. Before calling the children one needs to compute p_ℓ, p_r, L and R by scanning the list of unsorted intervals with buckets L and R. So it needs $O(n)$ time, where n is the number of intervals. Thus each inner node needs $O(n)$ time except for the calls for its children. Therefore the total running time of the algorithm is $O((2k^2)^k n)$. Since k is a constant it is $O(n)$. □

By slightly modifying the algorithm we can solve the similar dispersion problem on intervals where we can choose at most one point in each interval, as follows. (We can not choose two or more point in one interval in I.) If there is an interval, say $I' \in I$, containing both endpoints of an empty interval (i_{j-1}, i_j) in **Case** j, then we need to consider the following two more subcases. Case (L): The subinterval of I' appears only in L, with its right endpoint (i_{j-1}). Case (R): the subinterval of I' appears only in R, with its left endpoint (i_j) . Now the number of subproblems is at most $(4k^2)^k$, and the total running time of the algorithm is $O((4k^2)^k n)$. Since k is a constant it is $O(n)$.

Theorem 2. *One can solve the k-dispersion problem on intervals with the constraint that we can choose at most one point in each interval in $O(n)$ time even when the intervals are given unsorted on a line.*

3. k-dispersion for sorted intervals

If I is a set of sorted intervals on a line, and the coordinates of the endpoints of intervals are given as an array in which the coordinates are stored in the sorted order, then by slightly modifying the algorithm we can solve the dispersion problem in $O(\log n)$ time.

We can compute p_ℓ and p_r in $O(\log n)$ time using the array. We can also decide whether some point i is contained in some interval or not in $O(\log n)$ time by binary search on the array. Also instead of computing L explisitly, we can compute the leftmost interval in L and the rightmost interval in L in $O(\log n)$ time by binary search, and we can regard L as the intervals in I between those two intervals. Similar for R. Thus we can call each child with those information as arguments, instead of L and R. Now the running time is $O((2k^2)^k \log n)$, which is $O(\log n)$ since k is a constant.

Theorem 3. *One can solve the k-dispersion problem on intervals in $O(\log n)$ time when the intervals are given sorted on a line.*

4. Dispersion on Disks

Given a set D of disjoint disks and an integer $k \le |D|$, we wish to find k points in those disks so that the minimum distance between them is maximized. We can choose at most one point in each disk. The problem is called the dispersion problem on disks. The 1D version of the problem is the k-dispersion problem on intervals, which we have discussed in Section 2.

We need some notations. Let D^* be an optimal set of k points in D such that D^* contains at most one point in each disk, and $C(D^*)$ be the set of k center points of disks containing D^*. For a set S of points let $cost(S)$ be the minimum distance between two points in S. Let C be the set of center points of the disks in D, and S^* the set S of k points in C maximizing $cost(S)$. Let $d^* = cost(S^*)$.

For $k = n$ the problem is NP-hard, APX-hard, and a polynomial-time 0.707-approximation algorithm is known [13].

For $k \le n$ no results are known for the problem. We have the following lemma and two theorems.

Lemma 1. *Given a set C of n points and an integer $k \le n$ one can choose a set $S_A \subset C$ of k points in $O(n^2)$ time so that $cost(S_A) \ge cost(S^*)/2$.*

Proof. First we choose two points having the maximum distance in C. Let S_A be the set of the two points. Then repeatedly we append to S_A a point in $C - S_A$ having the maximum distance to S_A so that S_A has k points. We call this algorithm the greedy algorithm.

Consider the set S_D of k disks with radii $d^*/2$ having the centers $C(D^*)$. When we append a point to S_A there is a disk in S_D not containing a point in S_A. Now the disk has no point in S_A. Thus we can always find a point having no point in S_A within distance $d^*/2$. Therefore $cost(S_A) \ge d^*/2 = cost(S^*)/2$ holds. □

Theorem 4. *When D is a set of n disjoint disks with arbitrary radii, given an integer $k \le n$ one can choose a set S of k points in D in $O(n^2)$ time so that (1) no two point is contained in a disk in D and (2) $cost(S) \ge cost(D^*)/4$ holds.*

Proof. Now since D is disjoint $cost(C(D^*)) \ge cost(D^*)/2$ holds. Also $cost(S^*) \ge cost(C(D^*))$ holds. If we find a set S_A by Lemma 1 we have $cost(S_A) \ge cost(S^*)/2 \ge cost(C(D^*))/2 \ge cost(D^*)/4$. □

Theorem 5. *When D is a set of disjoint disks with uniform radii, say r, given an integer $k \le n$ one can find a set S of k points in D in $O(n^2)$ time so that $cost(S) \ge cost(D^*)/3$ holds.*

Proof. Now $cost(D^*) \ge cost(S^*)$. Since D is disjoint, $cost(D^*) - 2r \le cost(S^*)$ holds. Thus $cost(D^*) \le cost(S^*) + 2r$ holds. If we find a set S_A by Lemma 1 we have $cost(S_A) \ge cost(S^*)/2$ and so $cost(S^*) \le 2cost(S_A)$. Now $cost(D^*) \le 2cost(S_A) + 2r$. Therefore, since $cost(S_A) \ge 2r$, $cost(S_A)/cost(D^*) \ge cost(S_A)/(2cost(S_A) + 2r) = 1/(2 + 2r/cost(S_A)) \ge 1/3$ holds. □

See Fig. 3. The cost of optimal k points is 2 (See Fig. 3(a)), however the cost of k points computed by the greedy algorithm in the proof of Lemma 1 is 1. (See Fig. 3(b)). Thus there is an example for which the greedy algorithm computes a set S_A with $cost(S_A) = cost(S^*)/2$.

If we can choose any number of points in each disk, we have the following theorem for this version of the dispersion problem on disks. Let D^* be an optimal set of k points of the problem.

Theorem 6. *When D is a set of n disjoint disks with arbitrary radii, given an integer $k \le n$ and a real number $\epsilon \le 1$ one can choose a set S of k points in D in $O(n^2/\epsilon^4)$ time so that $cost(S) \ge cost(D^*)/(2(1+\epsilon))$ holds.*

Also one can choose a set S of k points in D in $O((n/\epsilon^2)^k)$ time so that $cost(S) \ge cost(D^*)/(1+\epsilon)$ holds.

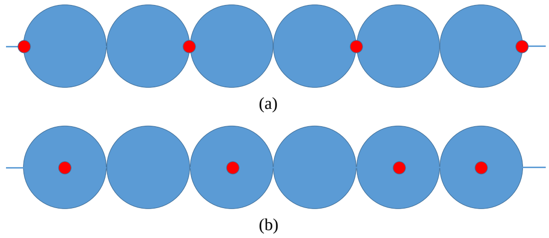

Fig. 3 An example with $cost(S_A) = cost(S^*)/2$. The radii of disks are 0.5.

Proof. Let r be the radius of the largest disk in D, and (x', y') the coordinate of the center of the largest disk. Since we can locate k points in the largest disk so that they evenly spaced on a line segment corresponding to the diameter, $cost(D^*) \ge 2r/(k-1) > 2r/k$ holds.

A point p located at (x, y) is a grid point iff $x = x' + (r\epsilon/ck)i$ and $y = y' + (r\epsilon/ck)j$ with some integers i and j. We will explain later the constant c which define the size of the cell of the grid.

Let S be the set of points that consists of (1) the centers of disks in D, and (2) the grid points contained in disks in D. Now $|S| \le n + (2r/(r\epsilon/ck))^2 n = n + 4(ck/\epsilon)^2 n$ holds, so $|S|$ is $O(n/\epsilon^2)$.

Let $S(D^*)$ be the set of points derived from D^* by choosing a nearest point in S for each point in D^*. We choose c large enough so that (1) $cost(S(D^*)) \ge cost(D^*)/(1+\epsilon)$ holds and (2) no two points in D^* have the common nearest point in S.

If we find a set S_A from S in $O(|S|^2)$ time by the greedy algorithm in the proof of Lemma 1, we have $cost(S_A) \ge cost(S^*)/2 \ge cost(S(D^*))/2 \ge cost(D^*)/(2(1+\epsilon))$.

If we find a set S^* in $O(|S|^k)$ time by a brute force algorithm we have $cost(S^*) \ge cost(S(D^*)) \ge cost(D^*)/(1+\epsilon)$. □

Thus this version of the dispersion problem on disks has a PTAS.

5. Conclusion

In this paper we have designed a simple algorithm to solve the k-dispersion problem on intervals. This is the first $O(n)$ time algorithm to solve the problem for any constant k.

Then we have shown, if intervals are given sorted on a line, by slightly modifying the algorithm, one can solve the problem in $O(\log n)$ time. This is the first sublinear time algorithm to solve the problem.

If disjoint intervals on a circle are given sorted an $O(n)$ time algorithm to solve the n-dispersion problem is known [5], [18]. Can we apply the method in this paper for the k-dispersion problem on disjoint intervals on a circle for any constant k?

We also have given some results for the k-dispersion problem on disjoint disks.

References

[1] T. Akagi and S. Nakano, Dispersion on the line, IPSJ SIG Technical Reports, 2016-AL-158-3 (2016).
[2] T. Akagi, T. Araki, T. Horiyama, S. Nakano, Y. Okamoto, Y. Otachi, T. Saitoh, R. Uehara, T. Uno, K. Wasa, Exact Algorithms for the Max-Min Dispersion Problem, Proc. of FAW 2018, LNCS 10823, pp.263-272 (2018).
[3] T. Araki and S. Nakano, Max-Min Dispersion on a Line, Journal of Combinatorial Optimization. Springer, (2020)
[4] C. Baur and S. P. Fekete, Approximation of geometric dispersion problems, Proc. of APPROX 1998, pp. 63–75 (1998).
[5] T. Biedl, A. Lubiw, A. M. Naredla, P. D. Ralbovsky and G. Stroud, Dispersion for Intervals: A Geometric Approach, Proc. of SOSA 2021 (2021)
[6] B. Birnbaum and K.J.Goldman, An improved analysis for a greedy remote-clique algorithm using factor-revealing LPs, Algorithmica, 50, pp. 42–59 (2009).
[7] S. Cabello, Approximation algorithms for spreading points, Journal of Algorithms, 62, pp.49–73 (2007).
[8] A. Cevallos, F. Eisenbrand and R. Zenklusen, Max-sum diversity via convex programming, Proc. of SoCG 2016, pp. 26:1–26:14 (2016).
[9] A. Cevallos, F. Eisenbrand and R. Zenklusen, Local search for max-sum diversification, Proc. of SODA 2017, pp. 130–142 (2017).
[10] B. Chandra and M. M. Halldorsson, Approximation algorithms for dispersion problems, J. of Algorithms, 38, pp. 438-465 (2001).
[11] Z. Drezner, Facility location: A Survey of Applications and Methods, Springer (1995).
[12] Z. Drezner and H.W. Hamacher, Facility Location: Applications and Theory, Springer (2004).
[13] A. Dumitrescu and M. Jiang, Dispersion in Disks, Theory of Computing Systems volume 51, pp.125–142 (2012).
[14] E. Erkut, The discrete p-dispersion problem, European Journal of Operational Research, 46, pp. 48–60 (1990).
[15] S. P. Fekete and H. Meijer, Maximum dispersion and geometric maximum weight cliques, Algorithmica, 38, pp. 501-511 (2004).
[16] J. Fiala, J. Kratochvil, A. Proskurowski, Systems of distant representatives, Discrete Appl. Math., 145, pp.306–36 (2005))
[17] R. Hassin, S. Rubinstein and A. Tamir, Approximation algorithms for maximum dispersion, Operation Research Letters, 21, pp. 133–137 (1997).
[18] S. Li and H. Wang, Dispersing Points on Intervals, Proc. of ISAAC 2016, Article No. 52 (2016) DOI: 10.4230/LIPIcs.ISAAC.2016.52
[19] 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).
[20] M. Sydow, Approximation guarantees for max sum and max min facility dispersion with parameterised triangle inequality and applications in result diversification, Mathematica Applicanda, 42, pp. 241–257 (2014).
[21] D. W. Wang and Y.-S. Kuo, A study on two geometric location problems, Information Processing Letters, 28, pp. 281–286 (1988).

