アルゴリズム　24－5
（1991. 11. 22）

平面上のn点からk個の最良点対間距離を求める解法

加藤　直樹*　　岩野　和生†

*　神戸商科大学　管理科学科
†　日本ＩＢＭ　東京基礎研究所

平面上にn点の集合$S=\{p_1,p_2,\ldots,p_n\}$が与えられているものとする。Sの中の点対によって定まる$\binom{n}{2}$個のL_p距離を小さい順に$d_1\le d_2\le\cdots\le d_{\binom{n}{2}}$とする。本論文では次の２つの問題を扱う。（１）$\binom{n}{2}$個の$L_p$距離のうち、最大のものから$k$個求める。（２）$\binom{n}{2}$個の$L_p$距離のうち、最小のものから$k$個求める。この２つの問題に対する$O(\min\{n^2, n+(k^{4/3}/\log^{1/3}k)\}\log n)$時間アルゴリズムを提案する。

Finding k Best Distances of n Points in the Plane

Naoki Katoh*　　Kazuo Iwano†

*　Department of Management Science,
Kobe University of Commerce
†　Tokyo Research Laboratory, IBM Japan

Given a finite set S of n points $p_1,p_2,\ldots,p_n$ in the plane, let $d_1\le d_2\le\cdots\le d_{\binom{n}{2}}$ be the L_p-distances determined by the pairs of points in S. This paper considers the following two problems:

Problem 1: For a given positive integer $k\le\binom{n}{2}$, enumerate k pairs of points which realize $d_{\binom{n}{2}-k+1},\ldots,d_{\binom{n}{2}}$.

Problem 2: For a given positive integer $k\le\binom{n}{2}$, enumerate k pairs of points which realize $d_1,\ldots,d_k$. We shall present an $O(\min\{n^2, n+(k^{4/3}/\log^{1/3}k)\}\log n)$ time algorithm for both problems.

1 Introduction

Given a finite set S of n points $p_1, p_2, \ldots, p_n$ in the plane, let $d_1 \leq d_2 \leq \cdots \leq d_{\binom{n}{2}}$ be the L_p-distances determined by the pairs of points in S. This paper considers the following two problems:

Problem 1 (Enumerating k Largest L_p-distances in the Plane) For a given positive integer $k \leq \binom{n}{2}$, enumerate k pairs of points which realize $d_{\binom{n}{2}-k+1}, \ldots, d_{\binom{n}{2}}$.

Problem 2 (Enumerating k Shortest L_p-distances in the Plane) For a given positive integer $k \leq \binom{n}{2}$, enumerate k pairs of points which realize $d_1, \ldots, d_k$.

In other words, problem 1 (resp. problem 2) is to report the k largest (resp. k shortest) distances among $\binom{n}{2}$ distances determined by $\binom{n}{2}$ pairs of points. We assume that for multiple pairs of equivalent distances, we may enumerate them in an arbitrary order.

Problem 2 has been studied by some researchers: Smid [12] presented an $O(n \log n)$ time algorithm for $k = O(n^{2/3})$ in general d-space for any L_p-metric, and Dickerson and Drysdale [5] recently gave an $O(n \log n + k \log n)$ time algorithm for L_2-metric. A closely related but slightly simpler version of Problem 2 is to select the k-th smallest distance d_k as well as a pair of points realizing it. For this problem, Chazelle [3] gave an $O(n^{9/5} \log^{4/5} n)$ time algorithm and Agarwal, Aronov, Sharir, and Suri [1] improved this time bound by giving an $O(n^{3/2} \log^{5/2} n)$ deterministic algorithm as well as $O(n^{4/3} \log^{8/3} n)$ randomized one. Both algorithms work for any dimension but only for L_2 metric. Salowe [11] extended Chazelle's result to general L_p-metric, and showed that this problem can be solved in $O(n \log^2 n)$ time for L_1 or L_∞ metric in the plane.

Regarding to Problem 1, to the authors' knowledge, however, no one has ever studied this problem. Furthermore, no one has ever studied Problem 2 for general L_p metric with $p \neq 2$ except Smid's result [12]. In this paper, we shall present an $O(\min\{n^2, n + (k^{4/3}/\log^{1/3} k)\} \log n)$ time algorithm for both problems. The techniques we use in our algorithms for both problems are (1) *k farthest neighbor Voronoi diagrams* for problem 1 and *k nearest neighbor Voronoi diagrams* for problem 2, and (2) the generalization of the technique developed by Frederickson and Johnson [7].

The k nearest neighbor Voronoi diagram (also called the *k-th order Voronoi diagram*) in the plane, $V_k(S)$, is a generalized notion of the *Voronoi diagram* in the plane, which is a subdivision of the plane into maximal regions, so that all points within a given region have the same set of k nearest neighbors. The k farthest neighbor Voronoi diagram, $V_k^f(S)$, is similarly defined. The first algorithm for computing $V_k(S)$ was presented by Lee [10], which requires $O(k^2 n \log n)$ time. Though his algorithm was given for L_2 metric, he remarked in [10] that the algorithm

can be generalized in a straightforward manner without changing time complexity to any L_p metric as well as to the k farthest neighbor Voronoi diagram. It has also been shown that, after obtaining $V_k(S)$ (or $V_k^f(S)$), k nearest (or farthest) neighbors of each point p_i can be computed within the time complexity of computing $V_k(S)$ (or $V_k^f(S)$). Later Edelsbrunner [6] gave an $O(k(n-k)\sqrt{n}\log n)$ time algorithm, and Chazelle and Edelsbrunner [4] presented an $O(n^2 + k(n-k)\log^2 n)$ time algorithm.

Notice that the following two trivial algorithms solve Problems 1 and 2. The first one sorts all pairs of distances and selects the desired k distances. Thus, it requires $O(n^2 \log n)$ time. Another naive approach to solve Problem 1 (*resp.* Problem 2) is to compute $\min\{n, k\}$ farthest (*resp.* $\min\{n, k\}$ nearest) neighbors of each point by constructing the $\min\{n, k\}$ farthest (*resp.* $\min\{n, k\}$ nearest) neighbor Voronoi diagram and to select k largest (*resp.* shortest) distances from the obtained $n \cdot \min\{n, k\}$ distances. We shall improve this $O(k^2 n \log n)$ time algorithm roughly by $O(n/k^{2/3})$ factor when $k \leq n^{3/2}$.

For this purpose, our algorithm for Problem 1 (*resp.* Problem 2) employs as a subroutine an $O(l^2 n \log n)$ time algorithm by Lee [10] for constructing $V_l^f(S)$ (resp. $V_l(S)$). Since the algorithms by [6] and [4] work only for L_2 metric and the algorithm by [4] requires $O(n^2)$ time for preprocessing, they are not appropriate for our purpose. Since the techniques we use for both problems are almost identical, we shall focus only on Problem 1 throughout this paper. Since the algorithm we shall present can be described independent of the underlying metric, we shall not explicitly indicate which metric we use in the algorithm.

Initially we apply Lee's algorithm for $l = \lceil 4k/n \rceil$. Then we have l farthest neighbors for each point p_i. From them we obtain $ln \approx 4k$ pairs of points in total. Then we select $2k$ largest distances. The key observation is that at least a half of points can be discarded in the further search for the desired k largest distances. Then, we again apply Lee's algorithm to the set of remaining points with the size of at most $n/2$ by letting $l = \lceil 4k/n' \rceil$, where n' is the number of remaining points. At each iteration, the algorithm computes $O(k)$ pairs of points that are stored in the candidate set, and halves the number of points at least a half and determines l as above for the next iteration. After iterating this process a certain number of times, the size of the remaining points becomes so small that directly computing all pairs of remaining points is less time consuming than applying Lee's algorithm. At this point, we apply such a direct method. At each iteration we obtain $O(k)$ canditates of pairs of points, and discard at least a half of the remaining points. We shall show that this approach significantly reduces the time complexity of the above naive approach.

We see that this technique can be regarded as a generalization of the one developed by Frederickson and Johnson [7] for finding the k-th smallest element in matrices with sorted rows and/or columns (see also [8], [9]). Selecting the k-th smallest element in an ordered set is a rather classical problem and is called the *selection problem*. The complexity of this problem is known to be linear in the size of an input set [2].

When the set is defined implicitly, this complexity is not necessarily optimal. For example, nontrivial time complexity was achieved for selecting the k-th smallest element in matrices with sorted rows and/or columns [7], [8], [9]. Chazelle [3] followed the techniques therein, and proposed a more general technique which enables us to efficiently solve several geometric selection problems including the following: (1) given two disjoint convex polygons with n vertices, compute k shortest bridges between them, (2) given n points in E^d, compute k shortest distances between pairs of points, (3) given n points in E^2, compute triangles formed by three points with k smallest areas. The technique we develop in this paper is different from the one by Chazelle [3], and can be viewed as another generalization of the one by Frederickson and Johnson [7].

The major contribution of this paper is not only to propose efficient algorithms for the above two problems but also to propose a general scheme that enables us to develop efficient algorithms for enumerating k best solutions for geometric optimization problems.

This paper is organized as follows. Section 2 gives the outline of the algorithm for problem 1. Section 3 describes the algorithm and analyzes the time complexity.

2 Outline of the Algorithm

As briefly mentioned in the previous section, our algorithm repeatedly applies Lee's algorithm by reducing the size of point set at least a half until the size becomes small enough so that the straightforward $O(n^2 \log n)$ time method is faster than Lee's algorithm. Suppose that the m-th iteration of Lee's algorithm obtains l_m farthest neighbors of each point for n_m remaining points. We use $l_m = \lceil 4k/n_m \rceil$ for $m \geq 1$.

At the m-th iteration, we obtain $l_m n_m \approx 4k$ pairs of points together with their distances. Among them, there may be two identical pairs of points, say (p_i, p_j) and (p_j, p_i); one pair (p_i, p_j) is obtained as one of l farthest neighbors of p_i and the other pair (p_j, p_i) as one of l farthest neighbors of p_j. For the technical reason, however, we regard them as distinct ones. Our algorithm proceeds by distinguishing these two identical pairs as if they are ordered pairs until the second stage of the algorithm. Therefore, the algorithm computes in the first stage $2k$ ordered pairs with $2k$ largest distances and then select from them the desired k unordered pairs with k largest distances in the second stage.

The first stage of the algorithm is carried out as follows. Initially it applies Lee's algorithm for $l_1 = \lceil 4k/n \rceil$ and $n_1 = n$ and computes $l_1 n_1 \approx 4k$ ordered pairs of points. This requires $O(\max\{n, (k^2/n)\} \log n)$ time. Then we select $2k$ ordered pairs with $2k$ largest distances from them. Other pairs are discarded because the distance for a discarded pair is not larger than the distance of any of the selected $2k$ pairs and it is not eligible for $2k$ largest distances we want to find.

An important observation here is that there are some points p_i such that some distance(s) between p_i and some other point(s) p_j is discarded, and that such points can be discarded from the succeeding search for the desired $2k$ largest distances. The reason is as follows: Suppose we are at the m-th iteration. For the discarded distance from p_i to p_j, p_j is one of l_m farthest neighbors of p_i (let p_j be the h-th farthest neighbor of p_i with $h \leq l_m$). Since each distance from p_i to h'-th ($h' \geq h$) farthest point of p_i is not larger than any distance in the $2k$ distances selected above. This implies that we need not search for l' farthest neighbors of p_i for $l' > l_m$. Thus the point p_i can be eliminated. Furthermore it can be shown that the number of such points is at least a half of of n_m since otherwise the number of remaining pairs of points is larger than $l_m \cdot (n_m/2)$, which is larger than or equal to $2k$ by the definition of l_m, which contradicts that the number of remaining pairs is $2k$.

For each discarded point p_i, if an ordered pair (p_i, p_j) belongs to the set of $2k$ selected pairs, the triplet $(d(p_i, p_j), i, j)$ is stored in the candidate set C. The set C is used to keep these triplets from which the desired $2k$ ordered pairs with $2k$ largest distances are eventually obtained at the end of the first stage.

Letting $l_{m+1} = \lceil 4k/n_{m+1} \rceil$ where n_{m+1} is the number of remaining points, we repeat the same process for the remaining point set. This requires $O((l_{m+1})^2 n_{m+1} \log n_{m+1})$ time. Repeating this process until the number of remaining points is less than $k^{2/3} \log^{1/3} k$. We then compute all pairs of the remaining points together with their distances in a straightforward manner because in this case this work is less expensive than applying Lee's algorithm. We then add the corresponding triplets to the candidate set C. Finally, $2k$ ordered pairs of points with $2k$ largest distances are selected from the set C by applying the linear time selection algorithm.

As mentioned before, the second stage is carried out to find k unordered pairs with k largest distances, which can be done in $O(k \log k)$ time as will be shown in the next section. The overall running time is shown to be bounded by $O(\min\{n^2, n + (k^{4/3}/(\log k)^{1/3})\} \log n)$. Therefore, this algorithm is much faster than the naive approach that was mentioned in Section 1.

3 The Algorithm

Based on the idea explained in the previous section, we shall describe the algorithm for selecting k largest distances of n points in the plane. Let $S = \{p_1, p_2, \ldots, p_n\}$ be the given n points. Let $F_l(p_i)$ be the set of k farthest points of p_i.

Algorithm (Finding k Largest L_p-Distances)

Step 0: $C := \emptyset$ and $m := 0$.

Step 1: $m := m+1, n_m := |S|$, and $l_m = \lceil 4k/n_m \rceil$. If $n_m \leq k^{2/3}(\log k)^{1/3}$, then go to step 4. Else apply Lee's algorithm to compute $F_{l_m}(p_i)$ for all $p_i \in S$. Let $D(p_i) := \{(d(p_i,p_j),i,j) \mid p_j \in F_{l_m}(p_i)\}$.

Step 2: (a) Let $D := \cup_{p_i \in S} D(p_i)$ and compute the subset D' of D with $2k$ largest distances by applying the linear time selection algorithm.

(b) For each $p_i \in S$, delete from $D(p_i)$ the triplets $(d(p_i,p_j),i,j)$ that are not selected in (a).

Step 3: (a) Let S' be the subset of S such that all l_m triplets in $D(p_i)$ are selected in D' in Step 2(a).

(b) Let $C := C \cup (\cup_{p_i \in S-S'} D(p_i))$ and discard the points in $S - S'$. Let $S = S'$ and return to Step 1.

Step 4: (compute all pairs of remaining points and their distances)

(a) For each $p_i \in S$, let $D(p_i) := \{(d(p_i,p_j),i,j) \mid i \neq j, p_j \in S\}$. Let $C := C \cup (\cup_{p_i \in S} D(p_i))$.

(b) Select $2k$ triplets with $2k$ largest distances in C.

Step 5: (second stage) Select k distinct unordered pairs of points with k largest distances and report them. Stop. □

As we discussed in the previous section, we iterate the loop of Steps 1 through 3 until the straightforward $O(n_m^2)$ time method becomes faster than Lee's algorithm to find l_m farthest neighbors of each point. Thus, we stop iterations when $n_m^2 \leq l_m^2 n_m \log n_m$, i.e., $n_m = O(k^{2/3}\log^{1/3} k)$.

Theorem 1 *The above algorithm correctly computes k pairs of points with k largest distances in $O(\min\{n^2, n + (k^{4/3}/(\log k)^{1/3}))\}\log n)$ time.*

Proof. The correctness of this algorithm is obvious from the discussion given so far. We then analyze the running time. Suppose that the loop of Steps 1 through 3 is iterated M times. Let n_m (*resp.* l_m) denote the number of points in S obtained in Step 1 (*resp.* the value of l) at the m-th iteration of the loop. When $k \leq n_1/4$, $l_1 = 1$ holds and the first iteration clearly requires $O(n\log n)$ time. In this case, Step 3 discards exactly $n_1 - 2k$ points and the number of remaining points is $2k$. Therefore, $k > n_m/2$ holds for all m with $m \geq 2$. From the discussion given in Section 2, $n_{m+1} \leq n_m/2$ follows. Therefore, the loop is iterated $M = O(\log(n/(k^{2/3}\log^{1/3} k)))$ times. The time required for the m-th loop with $m \geq 2$ is analyzed as follows. As shown in Lemma 2 below, Step 1 after $m \geq 2$ in total requires

$$O(\sum_{m=2}^{M} l_m^2 n_m \log n_m) = O(\sum_{m=2}^{M} (1/n_m)k^2 \log n) = O((k^{4/3}/\log^{1/3} k)\log n)$$

time. Step 2 requires $O(n_m l_m) = O(k)$ time. Step 3 also requires $O(k)$ time. Summing up these terms from $m = 2$ to $M = O(\log(n/(k^{2/3}\log^{1/3} k))$, we have

$$\begin{aligned} & O(k\log(n/(k^{2/3}\log^{1/3} k)) + (n/(k^{2/3}\log^{1/3} k)) \cdot (k^2/n)\log n) \\ = \; & O(k\log n + (k^{4/3}/\log^{1/3} k)\log n) \end{aligned}$$

time. Therefore the loop of Steps 1 through 3 requires in total $O(n \log n + (k^{4/3}/\log^{1/3} k) \log n)$ time.

When the loop of Steps 1 through 3 has not been executed at all (i.e., the case when $n \leq k^{2/3} \log^{1/3} k$), Step 4(a) requires $O(n^2)$ time, and Step 4(b) requires $O(n^2 \log n)$ time. Otherwise, Step 4(a) requires $O(k^{4/3}(\log k)^{2/3})$ time. When $l_1 = \lceil 4k/n \rceil \geq n^{3/2}/\log^{1/2} n$, Step 4(b) requires $O(n^2 \log n)$ time. Otherwise, since $O(k)$ triplets are added to C at every iteration and the loop of Steps 1 through 3 is iterated $O(\log n)$ times, Step 4(b) requires $O(k \log n)$ time. Step 5 is carried out as follows. First, for each triplet $(d(p_i, p_j), i, j)$ selected in Step 4(b), we swap i and j if $j > i$. After this, sort the $2k$ triplets in lexicographical order of the components of the triplets. Then we can choose in $O(k)$ time the desired k non-redundant unordered pairs of points with k largest distances. Summarizing the above discussion we establish the claimed time bound. □

Lemma 2 *$\sum_{m=2}^{M} 1/n_m = O(1/(k^{2/3} \log^{1/3} k))$ holds, where $M = \max\{m \mid n_m > k^{2/3} \log^{1/3} k\}$ is the number of iterations of the loop of Steps through 3.*

Proof. We first define two sequences $a_2, a_3, \cdots, a_M$ and $b_2, b_3, \cdots, b_{M'}$ as follows: let $a_m = 1/n_m$ for $m \geq 2$, $b_2 = a_2$, and $b_m = 2b_{m-1}$ for $m \geq 3$. Let $M' = \max\{m \mid 1/b_m > 2k^{2/3} \log^{1/3} k\}$. Let $f(m)$ be defined in such a way that $b_{f(m)-1} < a_m < b_{f(m)}$ for $m \geq 3$. Notice that since $2a_{m-1} \leq a_m \leq b_{f(m)}$, we have $b_{f(m)-1} < b_{f(m)}$ for all $m \geq 3$. We also have $b_{f(M)} \leq b_{M'}$. Therefore, we have

$$\sum_{m=2}^{M} a_m \leq \sum_{m=2}^{M} b_{f(m)} \leq \sum_{m=2}^{M'} b_m = O(2^{M'} a_2) = O(1/(k^{2/3} \log^{1/3} k))$$

□

4 Conclusion

This paper proposed an $O(\min\{n^2, n + (k^{4/3}/\log^{1/3} k)\} \log n)$ time algorithm for (1) enumerating k largest L_p-distances in the plane and (2) enumerating k shortest L_p-distances in the plane. To the authors' knowledge, no one has ever studied Problem 1, and we thus provided the first nontrivial algorithm for this problem. The idea behind the algorithm is general and may be applicable to other enumeration problems. For example, we can efficiently enumerate k shortest (*resp.* k largest) distances for the set of non-overlapping convex polygons or line segments if there exists an efficient algorithm for constructing k nearest (*resp.* k farthest) neighbor Voronoi diagrams for such geometric objects.

参考文献

[1] P.K. Agarwal, B. Aronov, M. Sharir, and S. Suri, Selecting Distances in the Plane, *Proceedings of the Sixth Annual ACM Symposium on Computational Geometry* 1990, 321-331

[2] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan, Time Bounds for Selection, *J. Computer and System Sciences* 7(1973), 448-461.

[3] B. Chazelle, New Techniques for Computing Order Statistics in Euclidean Space, *Proceedings of the First Annual ACM Symposium on Computational Geometry* 1985, 125-134

[4] B. Chazelle and H. Edelsbrunner, An improved algorithm for constructing kth-order Voronoi diagrams, *IEEE Transactions on Computers*, C-36 (11) (1987), 1349-1354.

[5] M. Dickerson and R.L. Drysdale, Enumerating k Distances for n Points in the Plane, *Proceedings of the Seventh Annual ACM Symposium on Computational Geometry* 1991.

[6] H. Edelsbrunner, Edge-skeltons in arrangements with applications, *Algorithmica*, 1 (1986), 93-109.

[7] G.N. Frederickson and D.B. Johnson, The complexity of selection and ranking in $X + Y$ and matrices with sorted columns, *J. Computer and System Sciences* 24 (1982), 197-208.

[8] G.N. Frederickson and D.B. Johnson, Generalized selection and ranking: sorted matrices, *SIAM J. on Computing*, 13 (1) (1984), 14-30.

[9] Z. Galil and N. Megiddo, A fast selection algorithm and the problem of optimum distribution of effort, *J. of ACM*, 26 (1979), 58-64.

[10] D.T. Lee, On k-nearest neighbor Voronoi diagrams in the plane, *IEEE Transactions on Computers*, C-31 (6) (1982), 478-487.

[11] J. Salowe, Selection problems in computational geometry, Ph.D Thesis, Department of Computer Science, Rutgers University, New Brunswick, New Jersey, 1987.

[12] M. Smid, Maintaining the minimal distance of a point set in less than linear time, *Universitat der Saarlandes* 06/90, 1990.