Two-Dimensional Range Search Based on the Voronoi Diagram

Takeshi Kanda[1] and Kokichi Sugihara[2]
[1, 2] Department of Mathematical Informatics,
Graduate School of Information Science and Technology, University of Tokyo
(Hongo 7-3-1, Bunkyo Ward, Tokyo 113-8685, JAPAN)
[1]kanda@simplex.t.u-tokyo.ac.jp, [2]sugihara@simplex.t.u-tokyo.ac.jp

Abstract: This paper studies the two-dimensional range search problem, and constructs a simple and efficient algorithm based on the Voronoi diagram. In this problem, a set of points and a query range are given, and we want to enumerate all the points which are inside the query range as quickly as possible. In most of the previous researches on this problem, the shape of the query range is restricted to particular ones such as rectangles and triangles, and the improvement on the worst-case performance has been pursued. On the other hand, the algorithm proposed in this paper is intended to accomplish a good average-case performance, and this performance is actually observed by numerical experiments. In these experiments, we compare the execution time of the proposed algorithm with those of other representative algorithms such as those based on the bucketing technique and the k-d tree. We can observe that our algorithm shows the best performance in almost all the cases.

Keywords: computational geometry, range search, side-trip facility search, Voronoi diagram

1 Introduction

The range search [4] [7] [8] [11] is one of the most fundamental and important geometric problems, and has been researched in the field of computational geometry. This problem has many applications such as computer graphics (for example, ray tracing and hidden surface removal), geographical information system, data-mining and so on. In this problem, a set S of N points and a query range R are given, and we want to enumerate all the points in S which are inside R. Considering that S is usually fixed, and query ranges arise many times; we are interested in preprocessing S in order to decrease the search time.

Most of the previous researches on this problem are restricted to the case where the shape of a query range R is orthogonal [1], half-planar [6], circular [3] [5] or spherical [2] [12]. In this paper, on the other hand, we propose an algorithm for solving the two-dimensional range search problem for a general shape of a query range R, by means of a simple technique using the Voronoi diagram. Furthermore, a good average-case performance is observed by numerical experiments.

In Section 2, we define the range search problem. In Section 3, we give an efficient algorithm for this problem, which is based on the Voronoi diagram. In Section 4, we experimentally show the performance of this algorithm. Lastly, conclusions are given in Section 5.

2 Range search problem

The **range search problem** in the two-dimensional space is defined as follows (See Figure 1):

> Given a set S of N points $\{P_n \mid n = 1, 2, \ldots, N\}$, preprocess the point set S, so that we can answer the query "Enumerate all the points in the set S inside the query range R?" quickly.

In many practical cases, the point set S is fixed once it is given, and a huge number of different query ranges are given one by one. A naive method of solving this problem is to check whether each point in S is inside R or not; however, this method requires as much as $O(N)$ time for each query. So, we are interested in preprocessing S which can decrease the search time.

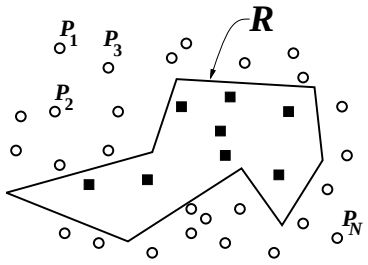

Figure 1. Range search problem. (Solid lines represent the query range, filled squares represent points to be reported, and empty circles represent other points.)

3 Range Search Based on the Voronoi Diagram

In this section, we give an algorithm of the two-dimensional range search using the Voronoi diagram. In Subsection 3.1, we review the Voronoi diagram. In Subsection 3.2, we consider the side-trip facility search, the algorithm of which forms the first step of that of the range search described in the following subsection. In Subsection 3.3, we construct the algorithm of the range search using the Voronoi diagram, and lastly, we demonstrate the validity of this algorithm in Subsection 3.4.

3.1 Voronoi Diagram

Given a set of N distinct points $\{P_i(x_i, y_i) \mid i = 1,\ 2,\ \ldots,\ N\}$ in the plane, we associate all locations in the plane with the closest member of the point set. The result is a tessellation of the plane into a set of regions associated with the members of the point set. This tessellation is called the **planar ordinary Voronoi diagram** generated by the point set [10]. The members of the point set are called **generators**. The region associated with a member P_i is denoted by $\mathcal{V}(P_i)$ and is called the **Voronoi polygon associated with** P_i. Vertices and edges in the Voronoi diagram are called **Voronoi vertices** and **Voronoi edges** respectively. We can restate the definition of each Voronoi polygon in mathematical terms as follows:

$$\mathcal{V}(P_i) = \bigcap_{j \neq i} \{P \mid d_i(P) \leq d_j(P)\} \ , \tag{1}$$

where $d_i(P)$ is the square distance between an arbitrary point $P(x, y)$ and a member $P_i(x_i, y_i)$ of the point set, that is to say,

$$d_i(P) = (x - x_i)^2 + (y - y_i)^2 \ . \tag{2}$$

Then, the Voronoi diagram generated by the point set $\{P_i\}$ is denoted by

$$\mathcal{V} = \{\mathcal{V}(P_i) \mid i = 1,\ 2, \ldots,\ N\} \ . \tag{3}$$

3.2 Side-trip Facility Search Problem

Before considering the algorithm solving the range search problem itself, let us start with considering the **side-trip facility search problem**. For points $Q_0, Q_1, \ldots, Q_M$ in the plane, let us denote by $\overline{Q_0Q_1 \ldots Q_M}$ the polygonal line consisting of M line segments $\overline{Q_0Q_1}$, $\overline{Q_1Q_2}$, ..., $\overline{Q_{M-1}Q_M}$. Here we consider the next problem (See Figure 2):

> Given a set S of N points $\{P_n \mid n = 1,\ 2,\ \ldots,\ N\}$, preprocess the point set S properly and answer one of the following queries quickly:
>
> **Type 1**: "Is there a point in the set S which is within the distance δ from the query polygonal line $\overline{Q_0Q_1 \ldots Q_M}$?"
>
> **Type 2**: "Which point in the set S is the closest to the query polygonal line $\overline{Q_0Q_1 \ldots Q_M}$?"

Type 3: "Enumerate all the points in the set S that are within the distance δ from the query polygonal line $\overline{Q_0Q_1 \ldots Q_M}$."

This problem may have some applications such as

(1) checking whether a circular robot can move along the polygonal line without hitting fixed circular obstacles (Type 1),

(2) finding the nearest facility from a car moving along polygonal road (Type 2),

(3) enumerating all houses at which the magnitude of noise coming from a road or a railroad is larger than the standard one (Type 3).

The algorithm for Type 1 and Type 2 is described in the following part of this subsection. Type 3 is not solved directly in this paper, but can be solved by regarding this problem as a special case of the range search problem. At first, we construct the Voronoi diagram of the input point set S in the preprocessing step. As shown in Subsection 3.3, the algorithm of the side-trip facility search problem will be used for each range search based on the Voronoi diagram as its first step.

Suppose that we are given the query polygonal line $\overline{Q_0Q_1 \ldots Q_M}$. Firstly, we determine which Voronoi polygon includes the starting point Q_0. This step is a special type of the **point location**, in that the given tessellation is restricted to be the Voronoi diagram. This can be done by tracing the Voronoi polygons from an arbitrary initial Voronoi polygon step by step. Furthermore, for many types of input points, the average-case time complexity can be constant by the joint use of a bucketing technique [9], which is used for finding a suitable starting point also in this paper.

In the main part of the side-trip facility search, we trace the Voronoi polygon along the given polygonal line $\overline{Q_0Q_1 \ldots Q_M}$ (See Figure 3). In the course of this step, whenever we enter a new Voronoi polygon, we calculate the distance between the given polygonal line and the input point corresponding to the Voronoi polygon we currently stop by. As for Type 1 of the side-trip facility search problem, if the distance is within δ, the search is over and the answer is "Yes". If the answer "Yes" is not obtained by the step in which we reach the Voronoi polygon including Q_M, the answer is "No". As for Type 2, on the contrary, when the distance is smaller than the minimum one we have ever calculated, we update the minimum distance and memorize the current nearest point. When we reach the Voronoi polygon including Q_M, we obtain the minimum distance and the point which gives it.

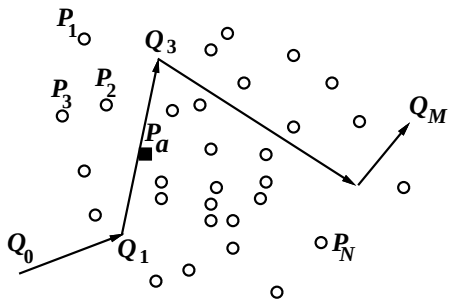

Figure 2. Side-trip facility search problem. (Bold arrows represent the query polygonal line; square P_a is the answer point for Type 2.)

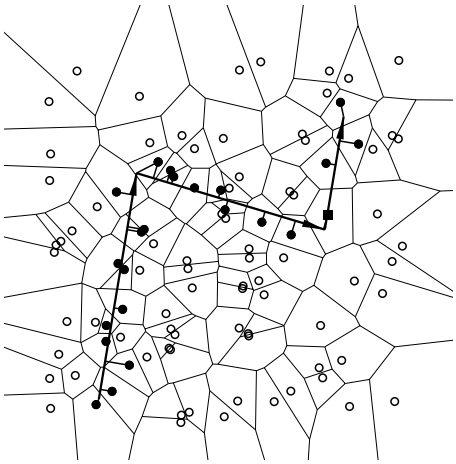

Figure 3. Side-trip facility search based on the Voronoi diagram. (Solid circles represent the input points whose distances to the query polygonal line are calculated; short lines represent such distances.)

3.3 Procedure of the Range Search Based on the Voronoi Diagram

In this subsection, we construct the algorithm of the range search based on the Voronoi diagram. Similar to the side-trip facility search, we firstly construct the Voronoi diagram of the input point set S as the preprocess for this algorithm. we also prepare an array $\{\mathtt{A}[1],\ \mathtt{A}[2],\ \ldots, \mathtt{A}[N]\}$, each element has already been set `False`. It means that each input point is not checked yet. In case that an input point P_n is checked, we set $\mathtt{A}[n]$ `True`.

As the preparatory step of each query, We prepare a queue or a stack Q, which stores all of the input points that has already been checked, but whose neighbours have not yet been checked completely. At first, Q is empty. Then, there are following four steps, Step 1, Step 2, Step 3 and Step 4, in each query. Step 4 is the preparation for the succeeding query.

Step 1 (Point location on the Voronoi diagram)

Choose an arbitrary end point, say Q_0, of the boundary of the query range R, and find the input point P_s nearest to Q_0.

Step 2 (Remaining part of the side-trip facility search) (See Figure 4.)

Starting with Q_0, execute the side-trip facility search along the boundary of the query range R until we reach Q_0 again.
For each input point P_n traced in this search, **begin**

If $A[n]$ is `False`, **begin**

Insert P_n into Q.
If P_n is inside R, report P_n.
Set $A[n]$ `True`.

end

end

Step 3 (Main part) (See Figure 5.)

While Q is not empty, **begin**

Delete one point P_n from Q.
For each point $P_{n'}$ neighbouring to P_n, **begin**

If $A[n']$ is `False`, **begin**

If $P_{n'}$ is inside R, report $P_{n'}$ and insert $P_{n'}$ into Q.
Set $A[n']$ `True`.

end

end

end

Step 4

Reset all of the elements $\mathtt{A}[1],\ \mathtt{A}[2],\ \ldots, \mathtt{A}[N]$ `False`.
(In order to avoid spending $\mathrm{O}(N)$ execution time at this step, we use another storage Q' which stores every input point checked so far. Thus, we can reset all the elements of A `False`, by just resetting the elements in Q'.)

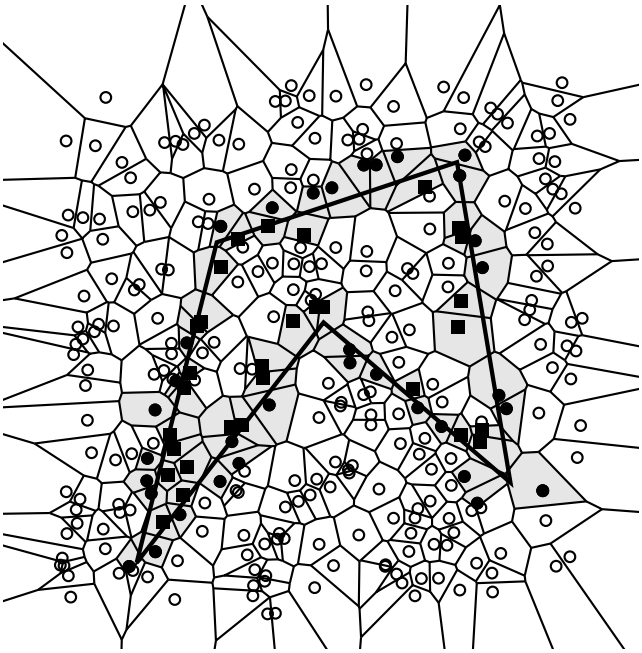

Figure 4. Second step of the range search based on the Voronoi diagram. (Bold lines represent the boundary of the query range R; solid circles and squares represent the points traced at Step 2; light gray polygons represent the Voronoi polygon associated with such points; solid squares represent the points found inside R in this step.)

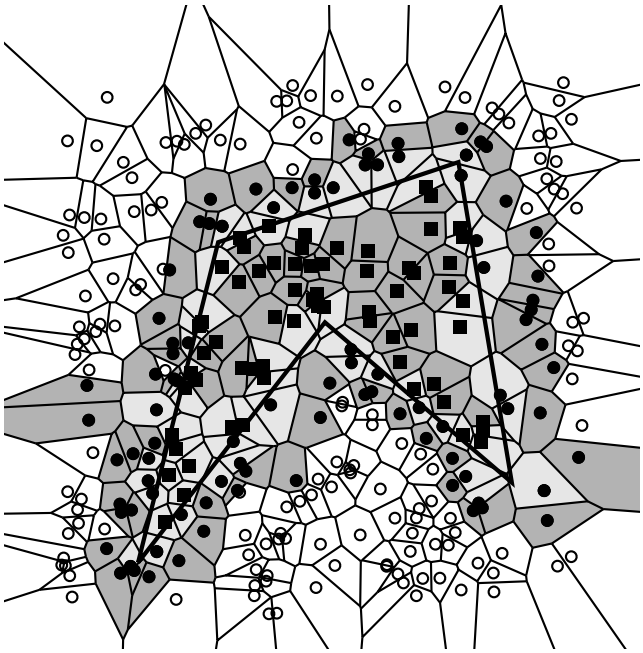

Figure 5. Third step of the range search based on the Voronoi diagram. (Solid circles and squares represent the points traced at Step 2 and Step 3; dark gray polygons represent the Voronoi polygon traced at Step 3; solid squares represent the points found inside R at both steps.)

3.4 Validity of the Range Search Based on the Voronoi Diagram

In this subsection, we demonstrate the validity of the algorithm described above.

Firstly, the algorithm finds all the point inside the query range R. In order to prove this, we assume that there exists a point that is inside R but is not reported at either step. Then, we can derive a contradiction in the following way (See Figure 5).

Proof

Let us start with classifying the input points into the following groups:

Type (I-1) (filled squares in Figure 5):
points which are inside R and are reported,
Type (I-2):
points which are inside R but are not reported,
Type (O-1) (filled circles in light gray polygons in Figure 5):
points which are outside R and are traced at Step 2,
Type (O-2) (filled circles in dark gray polygons in Figure 5):
points which are outside R, are not traced at Step 2, but are adjacent to at least one point which is traced at Step 2,
Type (O-3) (empty circles in Figure 5):
points which are outside R, are not traced at Step 2, and are not adjacent to any point which is traced at Step 2.

Assume that there exists a point P_m in Type (I-2). Then, P_m is adjacent only to points in Type (I-2) or Type (O-2) or Type (O-3); because otherwise, P_m would be checked and reported. Therefore, there exists at least one point in Type (I-2) which is adjacent to a point in Type (O-2) or Type (O-3) (note that it cannot be true that all the input points belong to Type (I-2)). Consequently, there exists at least one point which is inside R and is not traced at Step 2, but is adjacent to a point which is outside R and is not traced at Step 2. This is a contradiction (indeed, the Voronoi polygons associated with points traced at Step 2 cannot have a gap, because the polygons include the boundary of the query range R). We can conclude that any point in Type (I-2) never exists. Q. E. D.

Secondly, we estimate the execution time of the range search based on the Voronoi diagram. As far as the number of Voronoi polygons which are adjacent to each Voronoi polygon can be regarded as less than a certain constant, the execution time is proportional to the number of points traced at Step 2 and Step 3. These points are classified into three types: (I-1), which just represents the group of points inside R, and Type (O-1) and Type (O-2). Suppose that we are given the query range R, and that the density of the input points becomes larger and larger. Then, we can expect that the number of points contained in R grows much faster that the number of input points which belong to Type (O-1) or Type (O-2). Therefore, the total number of input points traced in the algorithm is $O(K)$, where K denotes the number of points inside R. On the other hand, we can expect that the execution time is constant with respect to the number of input point N, exclusive of that of Step 1. The time complexity evaluated as above is the best possible average-case performance.

4 Numerical Experiments

In this section, we show the merits of the algorithm proposed in this paper from a practical point of view. We compared the execution time of our algorithm (inclusive of the point location) with those of the following three other representative algorithms.

Bucket:
We use a simple bucketing technique. In this data structure, the whole square region where the

input points are distributed is divided into small square cells. In the following experiments, the region $[0,\ 1] \times [0,\ 1]$ is divided into 10×10 small square cells. Then we associate each of the input points one by one to the small cell which includes the input point.

At the first step of a query, we check whether each of the small cells intersects the query range. Then, only when the cell intersects the query range, we check whether each of the input points associated to the cell is inside the query range.

k-d Tree (Direct Use):
We use the k-d tree [1], one of the major tree structures used for geometrical search. Figure 6 is an example of the k-d tree. In this tree structure, the root node is associated with the whole region considered and one of the input points. All of the 2 child nodes, if any, are associated with smaller rectangular regions, both of which cover the region of the parent node without overlap. Each node is also associated with one of the input points within the corresponding region. The subdivision is done alternatingly by either the horizontal line or the vertical one passing through the input point corresponding to the current node. The direction of the division is decided by whether the depth of the current node is even or odd. The position of the division is selected so that the remaining input points are divided into 2 subsets with almost equal sizes.

In each query, we recursively traverse the tree to all of the leaves corresponding to the rectangular regions intersecting the query range, checking whether the input point associated to the rectangular region is inside the query range.

k-d Tree (After Triangulation):
We use the k-d tree after triangulating the query range. In other word, for each of the triangles generated by the triangulation, we adopt the same method mentioned above. Finally, we report all of the input points reported for either triangle so far.

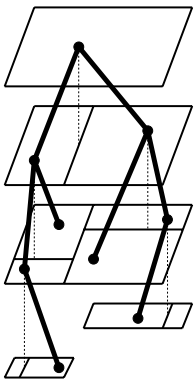

Figure 6. Examples of the k-d tree.

These experiments were done for two types of input points such as

Uniform Input Points (Left of Figure 7):
They are 10000 input points distributed uniformly in $[0,\ 1] \times [0,\ 1]$.

Clustered Input Points (Right of Figure 7):
They are 10000 input points with 100 clusters distributed uniformly in $[0,\ 1] \times [0,\ 1]$. Each of these clusters consists of 100 points distributed according to the normal distribution with standard deviation 1×10^{-2}.

In this experiment, the shapes of query ranges are also restricted to the three types of shapes such as

Square (Figure 8):
This is a square, whose edges are parallel to the x-axis or the y-axis.

Round Starred 100-gon (Figure 9):
This is a comparatively round starred 100-gons.

Acute Starred 100-gon (Figure 10):
This is a comparatively acute starred 100-gons.

Through the following experiments, all the data of execution times are averages of 100 or more executions.

Figure 7. Two types of input points tested in this paper, 100 input points distributed uniformly (left), 100 input points with 10 clusters distributed uniformly, each of which consists of 10 points distributed according to the normal distribution with standard deviation 3×10^{-2} (right).

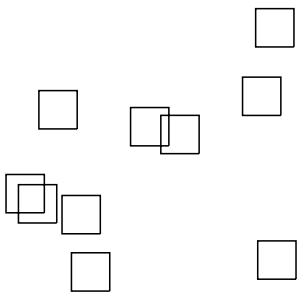

Figure 8. Ten examples of squares used as query ranges. The areas of these squares are constantly 0.01.

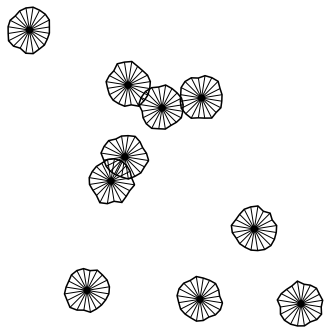

Figure 9. Ten examples of round starred polygons used as query ranges. These are comparatively round starred 20-gons whose area is constantly 0.01. Each of them is triangulated by the triangles, one of the vertices of which is the core of the starred 20-gon. In the experiments, 100-gons of the same type are used as examples of query ranges.

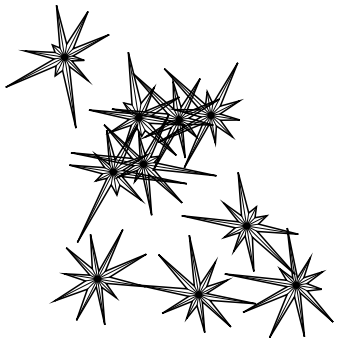

Figure 10. Ten examples of acute starred polygons used as query ranges. These are comparatively acute starred 20-gons whose area is constantly 0.01. Each of them is triangulated in the same manner. In the experiments, 100-gons of the same type are used as examples of query ranges.

Firstly, we fixed the number of input points N to 10000, and changed the area of the query range from 0.01, 0.02, ... to 0.10, corresponding to the expected number of reported points K being 100, 200, ... to 1000. The execution times are shown in the figures listed in Table 1. In these figures, the horizontal axes mean the areas of query squares and the vertical ones mean the execution times.

We can see the execution times of all the algorithms are linear in the area of query ranges, that is, the expected number of reported points K. Especially, as shown in Figure 11, in case that the number of angle of the query ranges is small, the algorithm based on the Voronoi diagram shows the best performance. Even in case that the number of angle of the query ranges is large, as shown in Figure 13 and Figure 15, as far as the area of the query ranges is small, the algorithm based on the Voronoi diagram still shows the best performance. If the query range is acute, that is, the length of the boundary is large, the algorithm using the k-d tree after triangulation is superior to that based on the Voronoi diagram. For the input points with clusters, the same tendency can be observed in Figure 12, 14 and Figure 16.

Table 1. Numbers of the figures showing the result of the numerical experiments which give the relation between K and the execution times.

Query Ranges \ Input Points	Uniform Input Points	Clustered Input Points
Square	Figure 11	Figure 12
Round Starred 100-gon	Figure 13	Figure 14
Acute Starred 100-gon	Figure 15	Figure 16

Secondly, we let the area of query ranges be so small that the expected number of reported points is 1, and let the number of input points N vary from 10000, 20000, ... to 100000. The execution times are shown in the figures listed in Table 2. In these figures, the horizontal axes mean the numbers of input points; the vertical ones mean the execution times.

As for the algorithm using the k-d tree and that based on the Voronoi diagram, we can see the execution times are nearly constant, not depending on the number of input points N, as shown in 17, 19 and Figure 21. We can conclude that, if the number of reported points is small enough, the algorithm based on the Voronoi diagram shows the best performance, for any types of query ranges tested in these experiments, and expectedly for many other types of query ranges, exemplified by Figure 18, 20 and Figure 22.

Table 2. Numbers of the figures showing the result of the numerical experiments which give the relation between N and the execution times.

Query Ranges \ Input Points	Uniform Input Points	Clustered Input Points
Square	Figure 17	Figure 18
Round Starred 100-gon	Figure 19	Figure 20
Acute Starred 100-gon	Figure 21	Figure 22

5 Conclusions

We have constructed a simple and efficient algorithm of solving the two-dimensional range search problem with respect to a general range shape. In this method, the Voronoi diagram for the input points is constructed in the preprocessing step, and is used to restrict the points to visit in the main step of the search. Then, for uniformly distributed input points and clustered input points, a good average-case performance is observed by numerical experiments; the execution time of this search is proportional to the area of query ranges, and little depends on the number of input points. Furthermore, we observed that the algorithm based on the Voronoi diagram shows the best performance in almost all the cases. It is only when the number of reported points is large and the number of angles of query ranges is also large, that the algorithm based on

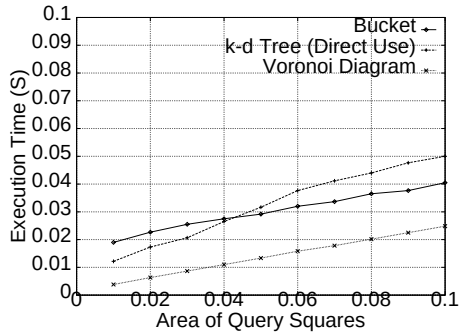

Figure 11. Relation between K and execution times of the range search for **Uniform Input Points** and **Squares**.

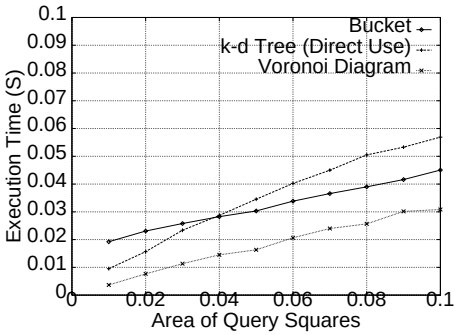

Figure 12. Relation between K and execution times of the range search for **Clustered Input Points** and **Squares**.

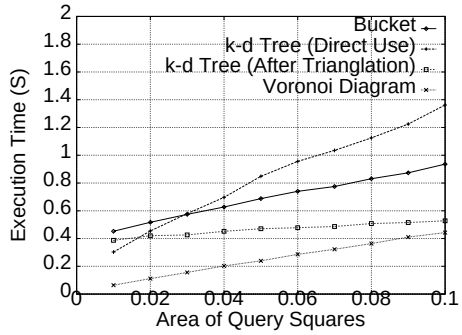

Figure 13. Relation between K and execution times of the range search for **Uniform Input Points** and **Round Starred 100-gons**.

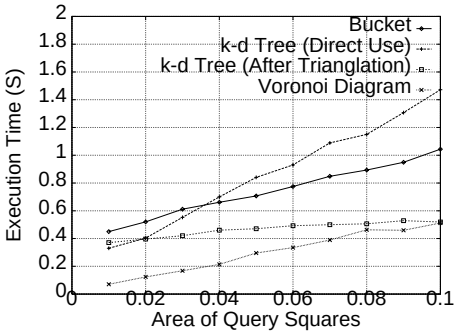

Figure 14. Relation between K and execution times of the range search for **Clustered Input Points** and **Round Starred 100-gons**.

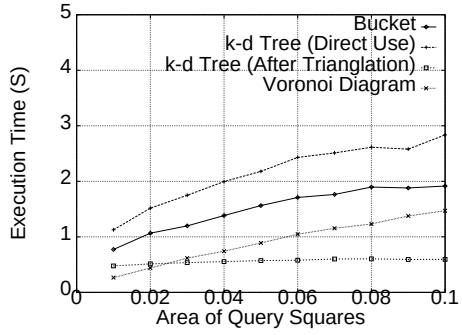

Figure 15. Relation between K and execution times of the range search for **Uniform Input Points** and **Acute Starred 100-gons**.

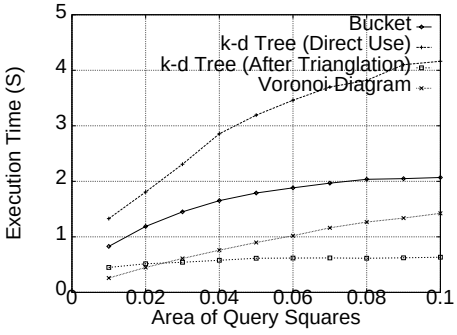

Figure 16. Relation between K and execution times of the range search for **Clustered Input Points** and **Acute Starred 100-gons**.

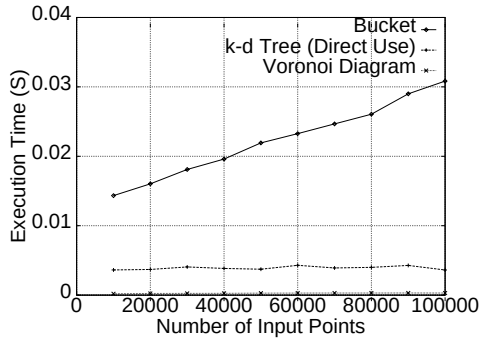

Figure 17. Relation between N and execution times of the range search for **Uniform Input Points** and **Squares**.

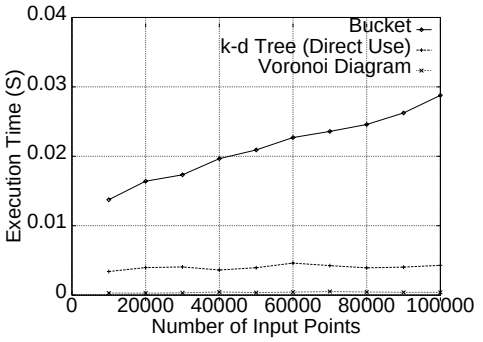

Figure 18. Relation between N and execution times of the range search for **Clustered Input Points** and **Squares**.

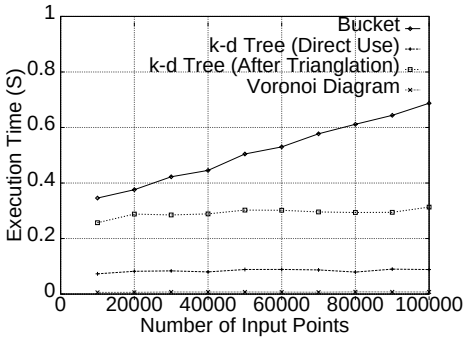

Figure 19. Relation between N and execution times of the range search for **Uniform Input Points** and **Round Starred 100-gons**.

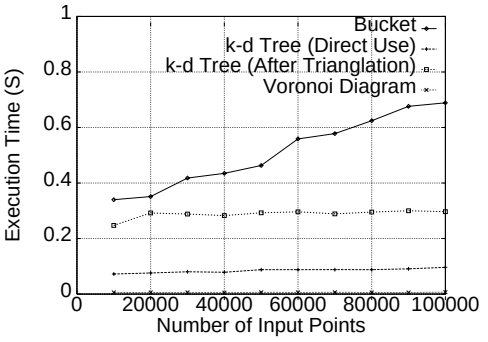

Figure 20. Relation between N and execution times of the range search for **Clustered Input Points** and **Round Starred 100-gons**.

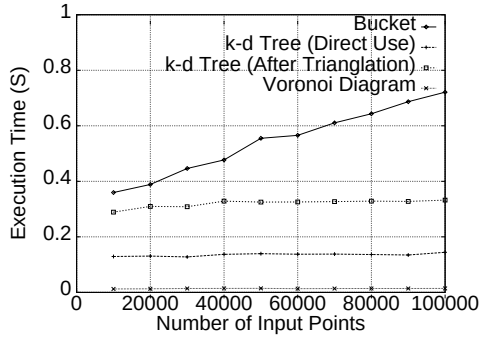

Figure 21. Relation between N and execution times of the range search for **Uniform Input Points** and **Acute Starred 100-gons**.

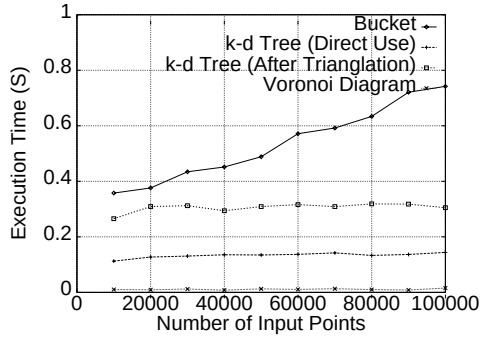

Figure 22. Relation between N and execution times of the range search for **Clustered Input Points** and **Acute Starred 100-gons**.

the Voronoi diagram is inferior to another algorithm, the one using the k-d tree after triangulation. So, from the practical point of view, the algorithm introduced in this paper can replace many of the other algorithms, as far as the dimension is two.

This paper is partially supported by the Grant-in-Aid for Scientific Research of the Japanese Ministry of Education, Science, Sports and Culture.

References

[1] J. L. Bentley: Multidimensional Binary Search Trees Used for Associative Searching, *Communications of the ACM*, Vol. 18, pp. 509–517, 1975.

[2] J. L. Bentley, D. F. Stanat, E. H. Williams, Jr.: The Complexity of Finding Fixed-radius Near Neighbors, *Information Processing Letters*, Vol. 6, No. 6, pp. 209–212, December 1977.

[3] J. L. Bentley, H. A. Maurer: A Note on Euclidean Near Neighbor Searching in the plane, *Information Processing Letters*, Vol. 8, No. 3, pp. 133–136, March 1979.

[4] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf: Computational Geometry — Algorithms and Applications —, Springer-Verlag, Berlin, Heidelberg, 1997.

[5] B. Chazelle, R. Cole, F. P. Prepareta, C. Yap: New Upper Bounds for Neighbor Searching, *Information and Control*, Vol. 68, pp. 105–124, 1986.

[6] H. Edelsbrunner, E. Welzl: Halfplanar Range Search in Linear Space and $O(n^{0.695})$ Query time, *Information Processing Letters*, Vol. 23, pp. 289–293, 1986.

[7] J. E. Goodman, J. O'Rourke: *Handbook of Discrete and Computational Geometry*, CRC Press, Boca Raton, New York, 1997.

[8] J. Matoušek: Geometric Range Searching, ACM Computing Survey, Vol. 26, No. 4, 1994.

[9] T. Ohya, M. Iri, K. Murota: Improvements of the Incremental Method for the Voronoi Diagram with Computational Comparison of Various Algorithms, *Journal of Operations Research Society of Japan*, Vol. 27, No. 4, pp. 69–97, 1984.

[10] A. Okabe, B. Boots, K. Sugihara, S.-N. Chiu: *Spatial Tessellations — Concepts and Applications of Voronoi Diagrams*, Second Edition, John Wiley & Sons, 2000.

[11] F. P. Preparata, M. I. Shamos: *Computational Geometry — An Introduction*, Springer-Verlag, New York, 1985.

[12] G. Yuval: Finding Near Neighbours in k-Dimensional Space, *Information Processing Letters*, Vol. 3, No. 4, pp. 113–114, March 1975.