Adaptive Point Location With Almost No Preprocessing in Delaunay Triangulations

Binhai Zhu
Department of Computer Science
Montana State University
Bozeman, MT 59717-3880
USA
Email: bhz@cs.montana.edu

***Abstract*—This paper studies adaptive point location in Delaunay triangulations with $o(n^{1/3})$ (and practically $O(1)$) preprocessing and storage. Given n pseudo-random points in a compact convex set C with unit area in two dimensions (2D) and the corresponding Delaunay triangulation, assume that we know the query points are clustered into k compact convex sets $C_i \subset C$, each with diameter $D(C_i)$, then we show that an adaptive version of the Jump&Walk method (which requires $o(n^{1/3})$ preprocessing) achieves average query bound $O(n^{\frac{1-4\delta}{3}})$ when in the preprocessing $\Theta(n^{\frac{1-4\delta}{3}})$ sample points are chosen within each C_i, where $D(C_i) = \Theta(\frac{1}{n^\delta})$ and $0 \leq \delta \leq 1/4$. Similar result holds in three dimensions (3D). Empirical results in 2D show that this procedure is 23%-350% more efficient than its predecessors under various clustered cases.**

Keywords: point location, Delaunay triangulation, jump-and-walk, probabilistic analysis.

I. Introduction

Point location is one of the classical problems in computational geometry, GIS, graphics and solid modeling. The theoretical problem is well studied in the computational geometry literature and several theoretically optimal algorithms have been proposed since early 1980s; see for example, Snoeyink's survey [26]. Practical point locations only received massive attention from computational geometers starting in 1990s [16], [13], [23], [14], [11]. All these works are somehow based on an idea due to Green and Sibson to use the "walkthrough" method to perform point location in Delaunay triangulation, a common data structure used in these areas. (We remark that similar ideas have been used in answering ray-shooting queries [2], [21].) In particular the Jump&Walk method of [16], [23] uses random sampling to select a good starting point to walk toward the destination while [13], [14] mix the "walkthrough" idea with some extra simple tree-like data structure to make the algorithm more general (for example, deal with arbitrary data [13] or handle extremely large input while bounding the query time [14]).

Although the algorithm of [16], [23] probably does not work very well for arbitrary input data and it is not as efficient as the one in, e.g., [14], it does have its own strength: its simplicity. It uses no preprocessing and is very easy to implement. Because of that, several famous software packages are using this algorithm, for example, the *Triangle* mesh generator [25], the *Qhull* package [`http://www.geom.umn.edu/software/qhull`] and the *X3D Grid Generation System* [27]. (In fact, in the X3D Grid Generation System, it took the builders just one weekend to update the point location subroutine in their system in 1996.) Because of these reasons, we investigate the Jump&Walk algorithm further in this paper, under some more realistic situations.

Theoretically, for pseudo-uniformly distributed points in a convex set C, in 2D Jump&Walk is known to have a running time of $O(n^{1/3})$ when the query point is slightly away from the boundary of C [16]. Similar result holds in 3D [23]. (The boundary condition in 2D was dropped by Devroye *et al*. in 2004 [15].) Certainly, an interesting question can be asked: *Is $\Omega(n^{1/(d+1)})$ the average query lower bound for point location in Delaunay triangulations of random points (within a general convex set), when no preprocessing is allowed?* Although we cannot answer this question definitely, in this paper we prove that if the query points are known to be clustered then we can adaptively change the Jump&Walk algorithm so that it runs in $o(n^{1/(d+1)})$ time for d=2,3. The new Adaptive Jump&Walk algorithm, similar to its predecessor, still retains the simplicity: with almost no preprocessing.

Studying geometric algorithms/data structures when the information on the query data is partially known is a new challenge for computational geometers. Arya *et al*. showed that if such information is known then it is possible to modify the known data structures to have an almost optimal average query bound for planar point location [3], [4], [6]. But in all these cases, either the preprocessing time or space is superlinear. In [5], Arya *et al*. presented a data structure which has expected linear space and achieves almost optimal expected query bound and they do not make any assumption on the subdivision and query data. (Our setting is different from those used by Ayra *et al.* and we assume that a Delaunay triangulation is given and we use $o(n^{1/3})$ preprocessing. Our assumption that the query points are clustered is realistic. Example 1, in a mesh smoothing procedure we update a particular region by inserting many Steiner points – all of them are clustered. Example 2, if we have a geographical map of Europe on-line, the region of London would be looked more during the 2012 Olympic Games.)

In general, point location deals with the following problem: given a set of disjoint geometric objects, determine the object containing a query point. The literature often restricts the objects to cells of subdivisions of geometric regions. In many applications, for example, in mesh generation and finite-element analysis (FEA), people focus even further on point location within Delaunay triangulations of dense points.

Delaunay triangulations. For completeness, we briefly mention the following definitions. Further details can be found in some standard textbooks like [24]. The *convex hull* of a finite point set X is the smallest convex set containing X. The convex hull of a set of $k+1$ affinely independent points in $\mathbf{R}^d$, for $0 \leq k \leq d$, is called a k-simplex; that is, a vertex, an edge, a triangle, or a tetrahedron, etc. If $k = d$, we also say the simplex is *full dimensional*. A *triangulation* $\mathcal{T}$ of X is a subdivision of the convex hull of X consisting of simplices with the following two properties: (1) for every simplex in $\mathcal{T}$, all its faces are also simplices in $\mathcal{T}$; (2) the intersection of any two simplices in $\mathcal{T}$ is either empty or a face of both, in which case it is again a simplex in $\mathcal{T}$. A *Delaunay triangulation* $\mathcal{D}$ of X is a triangulation in which the circumsphere of every full-dimensional simplex is empty, i.e., contains no points of X in its interior.

Point location by walking. The basic idea is straightforward; it goes back to early work on constructing Delaunay triangulations in 2D and 3D [19], [9]. Given a Delaunay triangulation $\mathcal{D}$ of a set X of n points in $\mathbf{R}^d$, and a query point q; in order to locate the (full-dimensional) simplex in $\mathcal{D}$ containing q, start at some arbitrary simplex in $\mathcal{D}$ and then "walk" from the center of that simplex to neighboring simplex "in the general direction" of the target point q. The underlying assumption is that the $\mathcal{D}$ is given by an internal representation allowing constant-cost access between neighboring simplices. The list of suitable data structures includes the 2D quad-edge data structure [20], the edge-facet structure in 3D [17], its specialization and compactification to the domain of 3D triangulations [22], or its generalization to d dimensions [10], etc.

Jump-and-Walk. In this algorithm, we modify the aforementioned method by jumping to a good starting point via random sampling on the data point set $\{X_1, X_2, \ldots, X_n\}$, i.e., we choose a set of $O(n^{1/(d+1)})$ simplices and walk from the one which is the closest to q. It is shown in [16], [23] that this algorithm takes expected $O(n^{\frac{1}{d+1}})$ time for $d = 2, 3$ (for $d = 3$ there is an extra $\log n / \log\log n$ factor). But it is not yet known whether $\Omega(n^{\frac{1}{d+1}})$ is the average lower bound for this problem.

Adaptive Jump-and-Walk. Given the Delaunay triangulation $\mathcal{D}$ of these n points $\{X_1, X_2, \ldots, X_n\}$, and a query point q which is known to be within one of the k regions $C_i (i = 1, ..., k)$ [1], the following procedure locates the simplex of $\mathcal{D}$ containing q, if such a simplex exists.

Preprocessing: For $i = 1$ to k do the following. Select m points $Y_{i1}, \ldots, Y_{im}$ at random and without replacement in C_i from $X_1, \ldots, X_n$. If a point X_l is not in C_i then repeat. In the end we generate m rample points in each of the k convex regions.

Query:

(1) Determine the index $j \in \{11, \ldots, 1m, \ldots, k1, \ldots, km\}$ minimizing the distance $d(Y_j, q)$. Set $Y = Y_j$.

(2) Locate the simplex containing q by traversing all simplices intersected by the line segment (Y, q).

Notice that we assume $k = O(1)$ and moreover; we assume C_i's can be described with $O(1)$ bytes so that whether a point lies in C_i can be determined in $O(1)$ time. Also, in practice instead of selecting random data points we choose random edges — which is easier for implementation [23]. Step (2), that is, the straight "walk," is easy to implement given the adjacency list implementation mentioned above.

In the next section, we will focus on proving the expected performance of the Adaptive Jump&Walk algorithm under the assumption that $X_1, ..., X_n$ are pseudo-uniformly distributed in a compact convex set C and the query points are known to be within k (k is a constant) compact convex subsets $C_i (i = 1, ..., k)$ of C.

Outline. The paper is organized as follows. In Section 2, we first prove the theoretical result. In Section 3, we present empirical results over randomly generated point sets ranging from $n = 10K$ to $50K$. Our tests confirm that the method with $O(1)$ preprocessing is more efficient than its the Jump&Walk method under several clustered cases. In Section 4, we conclude the paper with some open problems.

II. Theoretical analysis

We start by recalling some fundamental definitions. Let C be a compact convex set of $\mathbf{R}^2$ and let α and β be two reals such that $0 < \alpha < \beta$. We say that a probability measure P is an (α, β)-*measure* over C if $P[C] = 1$ and if we have $\alpha\,\lambda(S) \leq P[S] \leq \beta\,\lambda(S)$ for every measurable subset S of C, where λ is the usual Lebesgue measure. An $\mathbf{R}^2$-valued random variable X is called an (α, β)-random variable over C if its probability law $\mathcal{L}(X)$ is an (α, β)-measure over C. A particular and important example of an (α, β)-measure P is when P is a probability measure with density $f(x)$ such that $\alpha \leq f(x) \leq \beta$ for all $x \in C$. This probabilistic model was slightly general than the uniform distribution and we will loosely call it *pseudo-uniform* or *pseudo-random*.

We first prove a bound on the expected number of tries to generate the sample points in the preprocessing step. Throughout the paper, c_i, $i \geq 0$, denote positive constants depending upon the geometric properties of $C, C_1, ..., C_k$.

Theorem 1: Let C be a compact convex set with unit area in $\mathbf{R}^2$, C_i $(i = 1, ..., k)$ be compact convex subsets of C with diameter $D(C_i)$ and let $X_1, \ldots, X_n$ be n points drawn independently in C from an (α, β)-measure. The expected

[1] In practice, we can use $O(1)$ grids to partition $\mathcal{D}$. Over a large number of queries, C_i's can be identified approximately. This assumption is only for our proof.

number of tries to choose m points which lie in C_i from $X_1, \ldots, X_n$ is bounded by

$$c_0 \frac{km}{D(C_i)^2} \ .$$

Proof: Let $C, C_1, ..., C_k$ and $X_1, \ldots, X_n$ be as in Theorem 1. The probability that a point X_l lies in C_i is at least

$$\alpha \cdot area(C_i) = \alpha c_1 \pi D(C_i)^2 \ .$$

The expected number of tries to choose one point which lies in C_i from $X_1, \ldots, X_n$ is bounded by

$$\frac{1}{\alpha \cdot area(C_i)} = \frac{c_0}{D(C_i)^2} \ .$$

To generate m sample points in C_i, the expected number of tries is bounded by $c_0 \frac{m}{D(C_i)^2}$. ■

As $k = O(1)$, the preprocessing time and space is $O(m)$. As we require $m = \Theta(n^{\frac{1-4\delta}{3}}), 1 \leq \delta \leq 1/4$, to achieve the best query bound (see the theorem below), the preprocessing time and space are both $\Theta(n^{\frac{1-4\delta}{3}}), 1 \leq \delta \leq 1/4$, which is $o(n^{1/3})$. In practice, however, we only need to generate and store $O(1)$ sample edges in each cluster, so the preprocessing complexity is $O(1)$.

Below is the result on the expected running time of the Adaptive Jump-and-Walk algorithm, when applied on $\mathcal{D}$, the Delaunay triangulation of n random points in $\mathbf{R}^2$. Note that $d(.,.)$ denotes the Euclidean distance between points, $d(x, A)$ is $\inf_{y \in A} d(x, y)$ whenever A is a set.

Theorem 2: Let C be a compact convex set with unit area in $\mathbf{R}^2$, C_i $(i = 1, ..., k)$ be compact convex subsets of C with diameter $D(C_i)$ and let $X_1, \ldots, X_n$ be n points drawn independently in C from an (α, β)-measure. Assume that $C_i (i = 1, ..., k)$ is at distance of at least $c_6(\log n/n)^{1/2}$ from ∂C. If the query point q is independent of $X_1, \ldots, X_n$ and is known to be within $C_i (i = 1, ..., k)$, then the expected time of the simple algorithm given above is bounded by

$$c_2 m + c_3 \sqrt{\frac{n}{m}} D(C_i)^2 \ .$$

In particular, if $D(C_i) = c_4/n^\delta, 0 \leq \delta \leq 1/4$ and if $m = \lceil c_5 n^{\frac{1-4\delta}{3}} \rceil$ the expected time is $O(n^{\frac{1-4\delta}{3}})$.

The proof of the theorem rests on the following lemmas. In the following, Lemma 1 estimates the number of points of $X_1, \ldots, X_n$ which lie in C_i for some i.

Lemma 1: Let $C, C_1, ..., C_k$ and $X_1, \ldots, X_n$ be as in the Theorem. With probability $1/\phi$ $(\phi > 1)$, the number of points of $X_1, \ldots, X_n$ which lie in C_i, $N[X, C_i]$, is bounded by

$$\beta c_7 n \cdot area(C_i) = c_8 n \cdot D(C_i)^2 \ .$$

Proof: The expected number of points of $X_1, \ldots, X_n$ which lie in C_i, $E[X, C_i]$, is bounded by

$$\beta n \cdot area(C_i) = c_9 n \cdot D(C_i)^2 \ .$$

By Markov's inequality,

$$P\{N[X, C_i] > \phi E[X, C_i]\} \leq \frac{E[X, C_i]}{\phi E[X, C_i]} = \frac{1}{\phi},$$

where $\phi > 1$ is any positive constant. Therefore with probability at least $1 - \frac{1}{\phi}$, $N[X, C_i] \leq \phi E[X, C_i] \leq \phi \beta n \cdot area(C_i) = c_8 n \cdot D(C_i)^2$. ■

To estimate the number of triangles visited by the direct walk from Y to q, as in [16] we need the following lemma of [8] which is reorganized as follows.

Lemma 2: Let C, C_i be as in the Theorem and let $S[X, C_i]$ be the data points within C_i. If $\mathcal{L}$ is a fixed line segment of length $|\mathcal{L}|$ within C_i and if $\mathcal{L}$ is independent of $S[X, C_i]$, then the expected number of triangles or edges of the Delaunay triangulation for $X_1, \ldots, X_n$ crossed by $\mathcal{L}$ is bounded by

$$c_{10} + c_{15} |\mathcal{L}| \sqrt{N[X, C_i]} \ ,$$

which is $c_{10} + c_{12} |\mathcal{L}| D(C_i) \sqrt{n}$.

Proof: Without loss of generality, assume that q lies in C_i. Let Y^i be the sample point in C_i which is the closest to q. By the definition of Y, $d(Y, q) \leq d(Y^i, q)$. Following the results of [8], the expected number of triangles crossed by (Y, q) is smaller than or equal to that crossed by (Y^i, q). So from now on we focus on the segment (Y^i, q). To use Lemma 2 for a random line segment $\mathcal{L}$ (in C_i), we must first make sure that $\mathcal{L}$ is independent of the $N[X, C_i]$ points of $X_1, \ldots, X_n$ in C_i. This is done similarly to [16]. Let $\mathcal{D}_m$ be the Delaunay triangulation for data points $\{X_1, ..., X_n\} - \{Y_{i1}, \ldots, Y_{im}\}$. Then $\mathcal{L} = (Y^i, q)$, the line segment connecting Y^i and q, is independent of the data points $\{X_1, ..., X_n\} - \{Y_{i1}, \ldots, Y_{im}\}$. By Lemma 2, (Y^i, q) crosses an expected number of $c_{10} + c_{12} \mathbf{E} d(Y^i, q) D(C_i) \sqrt{n - m}$ edges in $\mathcal{D}_m$.

Notice that $d^2(Y^i, q)\pi$ is the probability contents of the circle at q of radius $d(Y^i, q)$, and is therefore distributed as the minimum of m i.i.d. (independently identically distributed) uniform $[0, c_{13} \cdot area(C_i)]$ random variables, which we call Z. Clearly, $\mathbf{E}\{Z\} = c_{13} \cdot area(C_i)/(m + 1)$. Let N denote the number of triangles in $\mathcal{D}$ crossed by (Y, q). Clearly $\mathbf{E}N$ is bounded by the number of triangles in $\mathcal{D}$ crossed by (Y^i, q) which is in turn bounded by the number triangles of $\mathcal{D}_m$ crossed by (Y^i, q) plus the sum S of the degrees of $Y_{i1}, \ldots, Y_{im}$ in the Delaunay triangulation $\mathcal{D}_m$. To see this, note that $\mathcal{L}$ either crosses a triangle without one of the Y_{il}'s as a vertex (in which case the triangle is identical in $\mathcal{D}$ and $\mathcal{D}_m$) or with one of the Y_{il}'s as a vertex. The total number of the latter kind of triangles does not exceed S. The expected value of S is, by symmetry, m times the expected degree of Y_{i1}, which is at most 6 by Euler's formula.

Following Lemma 1 and Lemma 2, we have

$$\begin{aligned}
\mathbf{E}N &\leq 6m + c_{10} + c_{15} \sqrt{c_{10} n D(C_i)^2 - m} \mathbf{E}\{d(Y^i, q)\} \\
&\leq 6m + c_{10} + c_{12} D(C_i) \sqrt{n} \mathbf{E}\{d(Y^i, q)\} \\
&\leq 6m + c_{10} + c_{12} D(C_i) \sqrt{n} \sqrt{\mathbf{E}\{d^2(Y^i, q)\}} \\
&= 6m + c_{10} + c_{12} D(C_i) \sqrt{n} \sqrt{\mathbf{E}\{(Z/\pi)\}} \\
&= 6m + c_{10} + c_{12} D(C_i) \sqrt{n} \sqrt{\{c_{14} \cdot area(C_i)/\pi(m + 1)\}} \\
&\leq 6m + c_{10} + c_{14} D(C_i) \sqrt{n} \sqrt{\{D(C_i)^2/(m + 1)\}} \\
&= 6m + c_{10} + c_{14} D(C_i)^2 \sqrt{n/(m + 1)} \ .
\end{aligned}$$

As it takes $O(1)$ time to visit a triangle from its neighbor, the expected running time T of the Adaptive Jump&

Walk is bounded by $\mathbf{E}\{T\} \leq O(m) + O(1) \cdot (6m + c_{10} + c_{14} D(C_i)^2 \sqrt{n/(m+1)})$, which implies that

$$\mathbf{E}\{T\} \leq c_2 m + c_3 D(C_i)^2 \sqrt{n/m}.$$

In particular, if $D(C_i) = c_4/n^\delta, 0 \leq \delta \leq 1/4$ and if $m = \lceil c_5 n^{\frac{1-4\delta}{3}} \rceil$ the expected time is $O(n^{\frac{1-4\delta}{3}})$. This concludes the proof of Theorem 2. ■

We remark that similar result holds for $d = 3$, with an extra $\log n / \log\log n$ factor, which corresponds to the expected maximum vertex degree in a random Delaunay triangulation, ignoring boundary effects [7]. The details are omitted.

III. Empirical results

In this section, we present some empirical results of the algorithm. In practice m should be a small constant as long as n is reasonably large (say, bounded by 2 million). The above observation immediately gives us a practical version of the algorithm which only stores $O(1)$ edges, which can be computed in $O(1)$ expected time. (For the ease of readers, we still call the algorithm Adaptive Jump&Walk.) When a query is performed we compute the sample edge which is the closest to the query point and then simply walk toward it, triangle by triangle. Throughout this section the input data (hence their Delaunay triangulation) are all within the unit square (0,0) and (1,1). We test the algorithm when the query points are clustered differently.

A. Small clusters

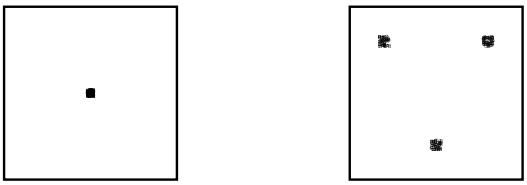

Fig. 1. 200 query points within one and three-clusters.

In this subsection we test this algorithm and compare it with Jump&Walk. We test it on two situations when the cluster regions are very small (conforming with conditions in Theorem 1). The 1-cluster is obtained by forcing all the query points to be within the square defined by $(0.48, 0.48)$ and $(0.52, 0.52)$. The 3-cluster contains three cluster squares defined by $(0.47, 0.17)$ and $(0.53, 0.23)$, $(0.77, 0.77)$ and $(0.83, 0.83)$ and $(0.17, 0.77)$ and $(0.23, 0.83)$. In Figure 1 we show two examples of 1-cluster and 3-cluster when there are 200 query points.

n	*J&W*	*Adaptive J&W*	Improvement
10K	44	14	214%
15K	51	12	325%
20K	56	12	367%
25K	58	13	346%
30K	58	13	346%
35K	57	13	338%
40K	71	18	294%
45K	75	13	477%
50K	75	21	257%

Table 1. Adaptive Jump&Walk vs Jump&Walk in 1-cluster.

Our empirical results are summarized in Table 1 and Table 2. For each n, we record the average cost (# of triangles visited plus the # of comparisons to find the edge which is the closest to q) over 1000 queries. The cost for generating sample edges in the new adaptive algorithm is always very small, hence the amortized cost of preprocessing (over 1000) is almost always 0. We compare the result with Jump&Walk. The last column in each table shows the percentage of improvement. In both of the tables, within each cluster we store 3 edges (sampled from a slightly larger copy — this is to play against a pathological situation when either the cluster is too small or when there are very few number of edges within the cluster). So for 1-cluster we store 3 edges while for 3-cluster we store 9 edges.

n	*J&W*	*Adaptive J&W*	Improvement
10K	41	21	95%
15K	54	23	135%
20K	55	21	162%
25K	59	25	136%
30K	63	34	85%
35K	66	26	154%
40K	66	26	154%
45K	74	34	118%
50K	74	34	118%

Table 2. Adaptive Jump&Walk vs Jump&Walk in 3-cluster.

B. Large clusters

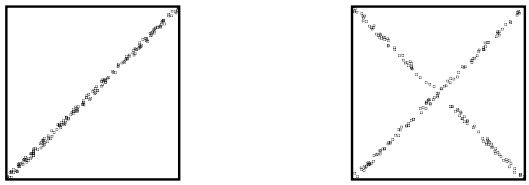

Fig. 2. 200 query points within the D-cluster and the X-cluster.

In this subsection we test the algorithm and compare it with Jump&Walk when the cluster regions are global and relatively larger relative to C (which do not conform with some of the conditions in Theorem 1). The D-cluster is obtained by enforcing all the query points to be close to the diagonal while the X-cluster is obtained by letting the query points be close to the two diagonals of the square. In Figure 2 we show two

examples of D-cluster and X-cluster when there are 200 query points.

n	$J\&W$	$Adaptive\ J\&W$	Improvement
10K	47	29	62%
15K	54	29	86%
20K	59	34	73%
25K	63	33	91%
30K	65	43	51%
35K	70	47	49%
40K	75	55	36%
45K	74	43	72%
50K	77	44	75%

Table 3. Adaptive Jump&Walk vs Jump&Walk in D-cluster.

The setting of experiments is almost the same as that in the previous subsection except that for D-cluster we store 12 edges while for X-cluster we store 18 edges (all from slightly fatter copies of the diagonals of the square). The comparisons are reported in Table 3 and Table 4 respectively.

n	$J\&W$	$Adaptive\ J\&W$	Improvement
10K	46	35	31%
15K	52	38	37%
20K	58	45	29%
25K	64	49	31%
30K	66	46	43%
35K	70	57	23%
40K	73	55	33%
45K	75	54	39%
50K	83	66	26%

Table 4. Adaptive Jump&Walk vs Jump&Walk in X-cluster.

IV. Discussions and closing remarks

The empirical results presented in the previous section show that the new adaptive algorithm (with $O(1)$ preprocessing in practice) is more efficient than its predecessor, the Jump&Walk method, when the query points are clustered. In fact, when we choose the sample edges from a fat copy of the cluster region, the smaller the cluster, the greater the improvement. In practice, if we know many of the query points are clustered, but we do not know the exact clustering regions, then we can partition the bounding box of C into some rectangles and over a large number (say 2000) of queries we can identify the rectangles containing a lot of query points — this will give us a good approximation for C_i's. Then, as in Section 3 we can store some random edges sampled from these rectangles to speed up the queries. This idea should also work if we have lots of queries, some of them are clustered and some of them are not. We remark that the method in [18], [13] should also work well for adaptive point location, but their methods require more preprocessing time and space than ours.

Although the Jump&Walk method has been studied extensively over the last 15 years, we still have some questions regarding this method. It would be interesting to know how the method performs on non-Delaunay triangulations used in practice. Recently, De Carufel *et al.* initiated some interesting work on Jump&Walk in well-shaped meshes [12]. It would be extremely interesting to know how 'well-shaped' meshes characterize the various practical datasets.

Finally, it would be interesting to know: *Is $\Omega(n^{1/(d+1)})$ the average query lower bound for point location in Delaunay triangulations of random points (within a non-degenerate convex set), when no preprocessing is allowed?* It is well-known that when enough ($\Theta(n)$ expected time and space) preprocessing is performed we can answer the queries in expected $O(1)$ time [1].

Acknowledgments

This research is partially supported by NSF under project DMS-0918034, by NSF of China under project 60928006, by the Shanghai Thousand Talents Program, and by the Open Fund of Top Key Discipline of Computer Software and Theory in Zhejiang Provincial Colleges at Zhejiang Normal University. The author would like to thank Sunil Arya for communicating his research results. Several anonymous reviewers provided valuable comments which are sincerely appreciated.

References

[1] T. Asano, M. Edahiro, H. Imai, M. Iri, and K. Murota. Practical use of bucketing techniques in computational geometry. In G. T. Toussaint, editor, *Computational Geometry*, pages 153–195. North-Holland, Amsterdam, Netherlands, 1985.

[2] B. Aronov and S. Fortune. Average-case ray shooting and minimum weight triangulations. In *Proceedings of the 13th Symposium on Computational Geometry*, pages 203–212, 1997.

[3] S. Arya, S.W. Cheng, D. Mount and H. Ramesh. Efficient expected-case algorithms for planar point location. In *Proceedings of the 7th Scand. Workshop on Algorithm Theory*, pages 353–366, 2000.

[4] S. Arya, T. Malamatos and D. Mount. Nearly optimal expected-case planar point location. In *Proceedings of the 41th IEEE Symp on Foundation of Computer Science*, 2000.

[5] S. Arya, T. Malamatos and D. Mount. A simple entropy-based algorithm for planar point location. In *Proceedings of the 12th ACM/SIAM Symp on Discrete Algorithms*, Jan, 2001.

[6] S. Arya, T. Malamatos and D. Mount. Entropy-preserving cuttings and space-efficient planar point location. In *Proceedings of the 12th ACM/SIAM Symp on Discrete Algorithms*, Jan, 2001.

[7] M. Bern, D. Eppstein, and F. Yao. The expected extremes in a Delaunay triangulation. *International Journal of Computational Geometry & Applications*, 1:79–91, 1991.

[8] P. Bose and L. Devroye. Intersections with random geometric objects. *Comp. Geom. Theory and Appl.*, 10:139–154, 1998.

[9] A. Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24:162–166, 1981.

[10] E. Brisson. Representing geometric structures in *d* dimensions: Topology and Order. *Discrete & Computational Geometry*, 9(4):387–426, 1993.

[11] P. de Castro and O. Devillers. A pedagogic JavaScript program for point location strategies. In *Proceedings of the 27th Symposium on Computational Geometry*, pages 295-296, 2011.

[12] J-L. De Carufel, C. Dillabaugh and A. Maheshwari. Point location in well-shaped meshes using Jump and Walk. In *Proceedings of the 23rd Canadian Conf. on Computational Geometry (CCCG'11)*, August, 2011.

[13] O. Devillers. Improved incremental randomized Delaunay triangulation. In *Proceedings of the 14th Symposium on Computational Geometry*, pages 106–115, 1998.

[14] L. Devroye, C. Lemaire and J-M. Moreau. Fast Delaunay point location with search structures. In *Proceedings of the 11th Canadian Conf on Computational Geometry*, pages 136–141, 1999.

[15] L. Devroye, C. Lemaire and J-M. Moreau. Expected time analysis for Delaunay point location. *Comp. Geom. Theory and Appl.*, 29(2):61-89, 2004.

[16] L. Devroye, E. P. Mücke, and B. Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, Special Issue on Average Case Analysis of Algorithms, 22(4):477-482, 1998.

[17] D. P. Dobkin and M. J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica*, 4(1):3–32, 1989.

[18] M. T. Goodrich, M. Orletsky, and K. Ramaiyer. Methods for achieving fast query times in point location data structures. In *Proceedings of Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, pages 757–766, 1997.

[19] P. J. Green and R. Sibson. Computing Dirichlet tessellations in the plane. *The Computer Journal*, 21:168–173, 1978.

[20] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[21] J. Hershberger and S. Suri. A pedestrian approach to ray shootings: shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

[22] E. P. Mücke. Shapes and Implementations in Three-Dimensional Geometry. Ph.D. thesis. Technical Report UIUCDCS-R-93-1836. Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1993.

[23] E. P. Mücke, I. Saias and B. Zhu. Fast randomized point location without preprocessing in two and three-dimensional Delaunay triangulations. *Comp. Geom. Theory and Appl.*, Special Issue for SoCG'96, 12(1/2):63-83, 1999.

[24] F. P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[25] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Proceedings of the First ACM Workshop on Applied Computational Geometry*, pages 124–133, 1996.

[26] J. Snoeyink. Point location. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 559–574. CRC Press, Boca Raton, 1997.

[27] H. Trease, D. George, C. Gable, J. Fowler, E. Linnbur, A. Kuprat and A. Khamayseh. *The X3D Grid Generation System.* In *Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Field Simulations*, 239–244, 1996.