Computing the Umbrella Neighbourhood of a Vertex in the Delaunay Triangulation and a Single Voronoi Cell in Arbitrary Dimension

Tyler H. Chang*, Layne T. Watson*†‡, Thomas C. H. Lux*, Sharath Raghvendra*,
Bo Li*, Li Xu§, Ali R. Butt*, Kirk W. Cameron*, and Yili Hong§
*Dept. of Computer Science
†Dept. of Mathematics
‡Dept. of Aerospace and Ocean Engineering
§Dept. of Statistics
Virginia Polytechnic Institute and State University
Blacksburg, Virginia, 24061
Email: thchang@vt.edu

***Abstract*—Delaunay triangulations and their geometric dual Voronoi diagrams are carefully studied topics in computational geometry and have numerous applications spanning fields such as physics, engineering, geographic information systems, and computer graphics. There are numerous efficient algorithms for computing both in two and three dimensional real space, but for higher dimensional space, the computational complexity grows exponentially. For many applications, it is only necessary to compute the star of simplices incident to (often called the umbrella neighbourhood of) a single vertex of the Delaunay triangulation, or equivalently, the Voronoi cell of a point. In practice, this may be a relatively small subset of the total Delaunay triangulation or Voronoi diagram. In this paper, an algorithm is proposed for computing the umbrella neighbourhood of a single vertex in the Delaunay triangulation.**

1. Introduction

Delaunay triangulations and Voronoi diagrams are fundamental constructs from computational geometry. Both are widely used for mesh generation, interpolation, and topological shape approximation, in fields such as physics, engineering, geographic information systems, and computer graphics. They are also geometric duals of each other, and therefore an algorithm that computes either one is easily extended to compute the other.

A d-dimensional tessellation of a convex region $X \subseteq \mathbb{R}^d$ is a division of X into closed (typically convex) sets called cells that are disjoint except along shared boundaries and whose union is X. Given a set of points $P \subset \mathbb{R}^d$, a d-dimensional triangulation is a tessellation of the convex hull of P, denoted $CH(P)$, where each cell is a d-simplex with vertices in P. The Voronoi diagram (also called the Dirichlet tessellation) of a finite set of points $P \subset \mathbb{R}^d$ is a specific tessellation of the entire space $\mathbb{R}^d$, derived from the nearest neighbor relationship on P. The Delaunay triangulation is then a specific triangulation of P derived from the Voronoi diagram.

In two dimensions, there are numerous efficient methods for computing both Voronoi diagrams [1] and Delaunay triangulations [2]. However, in higher-dimensional spaces, the computational complexity has been shown to grow exponentially [3]. In order to save on computational time and space complexity, this paper considers applications where only a small number of cells from the Voronoi diagram or a small subset of the Delaunay triangulation (specifically the umbrella neighbourhood of a point) is needed.

The paper will proceed as follows. Section 2 provides relevant definitions and a short summary of related work and challenges. Section 3 contains a detailed description of the operations performed in the proposed algorithm, and the necessary data structures. Section 4 introduces the proposed algorithm along with an analysis of its time and space complexity. Section 5 briefly mentions several degeneracy and numerical stability issues that could arise and how they could be addressed. Section 6 briefly describes a Fortran implementation of the algorithm and presents an empirical analysis of its run time. Section 7 concludes this paper and outlines future work.

2. Background

2.1. Definitions

The Voronoi diagram of a finite set of points $P \subset \mathbb{R}^d$ is defined in terms of a nearest neighbor relationship using the Euclidean distance metric.

Definition 1. *The Voronoi diagram $VD(P)$ of a set of n points $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ is the unique tessellation of $\mathbb{R}^d$ into n cells $VD(P) = \{V_1, \ldots, V_n\}$ such that for all $V_i \in VD(P)$ and $x \in \mathbb{R}^d$, $x \in V_i$ if and only if $\|p_i - x\|_2 \leq \|p_j - x\|_2$ for all $j \neq i$.*

978-1-5386-6133-8/18/$31.00

Notice that any point x on the boundary between k adjacent Voronoi cells V_{i_1}, ..., V_{i_k} (where $k \geq 2$) must satisfy

$$\|p_{i_1} - x\|_2 = \ \cdots \ = \|p_{i_k} - x\|_2. \tag{1}$$

So, the boundary between k adjacent Voronoi cells is the solution space to a system of $k-1$ linear equations. Unless these equations are dependent (which occurs if and only if all k points lie in some $(k-3)$-sphere) the solution space for a system of $k-1$ linear equations is a $(d-k+1)$-dimensional linear manifold. Therefore, each Voronoi cell is a d-polytope with vertices satisfying (1) for $k \geq d+1$ and points p_{i_1}, ..., p_{i_k}.

Note that as described above, any $k \geq d+1$ points defining a Voronoi vertex v must be some distance r from v, and the open ball $B(v,r) = \{x \mid \|x-v\|_2 < r\}$ must satisfy $B \cap P = \emptyset$. Furthermore, these k points must lie on the topological boundary of $B(v,r)$, which is the $(d-1)$-sphere $C(v,r) = \{x \mid \|x-v\|_2 = r\}$. Conversely, note that any $d+1$ affinely independent points in $\mathbb{R}^d$ define a unique $(d-1)$-sphere $C(v,r)$ that passes through these points, and if $B(v,r)$ is empty, then v must be a Voronoi vertex.

The Delaunay triangulation is defined as the geometric dual of the Voronoi diagram [4]. To obtain a Delaunay triangulation from the Voronoi diagram of P, take the set of d-simplices defined by the $d+1$ closest points to each Voronoi vertex (see Figure 1). By definition of a Voronoi vertex, there will always be at least $d+1$ points equidistant from each vertex. If there are more than $d+1$ points equidistant from one or more Voronoi vertices, then the division of those points into two or more space-filling d-simplices is arbitrary, and any Delaunay triangulation of P is not unique.

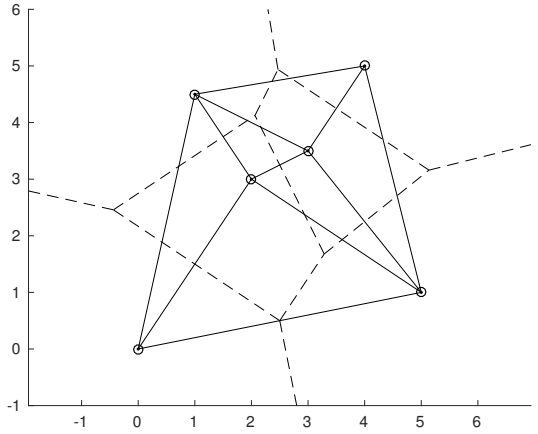

Figure 1. The Voronoi diagram of a set of points P in $\mathbb{R}^2$ with the Delaunay triangulation overlayed.

Recall that each Voronoi vertex v is the center of a $(d-1)$-sphere $C(v,r)$ through the $d+1$ or more points defining v, such that $B(v,r) \cap P = \emptyset$. This is often referred to as the *empty circumsphere property*. Formalizing this property, the following alternative definition of a Delaunay triangulation is generally preferred. See Figure 2 for a visual.

Definition 2. *A Delaunay triangulation $DT(P)$ of a finite set of points $P \subset \mathbb{R}^d$ is any triangulation of P such that for each d-simplex $S \in DT(P)$, the $(d-1)$-sphere $C(v,r)$ circumscribing S satisfies $B(v,r) \cap P = \emptyset$.*

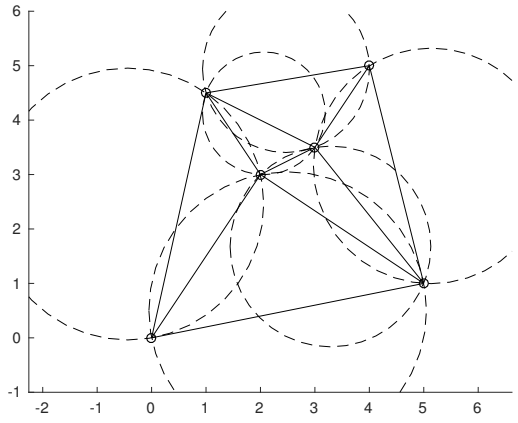

Figure 2. The empty circumsphere property demonstrated in $\mathbb{R}^2$.

Note that if all $p \in P$ are contained in some lower-dimensional linear manifold (or equivalently, if $CH(P)$ has zero volume), then all Voronoi edges extend without bound, and there are no Voronoi vertices. Therefore, $DT(P)$ (and in general, any full-dimensional triangulation of P) does not exist. It should be noted that for any set of n points in $\mathbb{R}^d$, if $n < d+1$ then all of the points lie in a $(n-1)$-dimensional linear manifold trivially and no triangulation can exist.

Except in degenerate cases, there will be exactly $d+1$ points on every $(d-1)$-sphere, and $DT(P)$ is unique. Generally, if $n \geq d+1$, it can be assumed that $DT(P)$ exists and is unique. Since the existence and uniqueness of $DT(P)$ is generally presumed, it is customary to reference *the* Delaunay triangulation of a set of points. This can be formalized with the following definition.

Definition 3. *A set of points P in $\mathbb{R}^d$ is said to be in general position if P does not lie in some $(d-1)$-dimensional linear manifold, and if no $d+2$ points in P lie on the same $(d-1)$-sphere.*

Note that the circumcenter of every Delaunay simplex is a Voronoi vertex and vice versa. Also, recall that v is a Voronoi vertex if and only if the sphere $C(v,r)$ passing through the nearest $d+1$ affinely independent points satisfies $B(v,r) \cap P = \emptyset$. This leads to the following useful equivalence.

Observation 1. *A set of $d+1$ points $s \subseteq P$ is the set of vertices of a d-simplex $S \in DT(P)$ (whose circumcenter is a Voronoi vertex) if and only if s defines a $(d-1)$-sphere $C(v,r)$ with $B(v,r) \cap P = \emptyset$.*

For a corollary, consider the "star" of simplices in $DT(P)$ that share some vertex $\hat{p} \in P$. This subset of $DT(P)$ is often called the *umbrella neighbourhood* of $\hat{p}$. From Observation 1, it follows that the set of circumcenters for all the simplices in the umbrella neighbourhood of $\hat{p}$ is exactly the set of vertices of the Voronoi cell containing $\hat{p}$.

2.2. Applications and Properties

The Voronoi diagram allows for rapid nearest neighbor queries, among many other applications. The natural neighbor interpolant is defined as a convex combination of nearby Voronoi cells as described in [5], and this usage is extended to define an entire finite element mesh in [6]. Voronoi diagrams have also been used in the context of topological shape approximations for constructing skeletons [7].

The Delaunay triangulation is most widely used to define a piecewise linear interpolant and corresponding unstructured mesh [8]. It has been shown that the Delaunay interpolant and mesh are in many ways optimal with respect to other simplex based interpolation schemes and meshes [9]. The Delaunay triangulation also sees wide use in topological shape approximations as a means to compute α-shapes and α-complexes [10].

It should be noted that while many applications require knowledge of the entire Voronoi diagram or Delaunay triangulation, many other applications can be achieved by computing only a small number of Voronoi cells or the umbrella neighbourhoods of relatively few points. For example, in [11] the volumes of Voronoi cells are used to compute the porosity in various regions of a pebble-bed nuclear reactor. In the context of multiobjective optimization, the Delaunay umbrella neighbourhood of each point is used to approximate the relative isolation of points on the pareto front [12].

2.3. Related Work and Limitations

Efficient solutions exist for computing both Voronoi diagrams [1] and Delaunay triangulations [2] in two-dimensions, running in $\mathcal{O}(n \log n)$. However, in $\mathbb{R}^d$, the worst case size of the Delaunay triangulation and total number of Voronoi vertices is known to be $\mathcal{O}(n^{\lceil d/2 \rceil})$ [3]. Even in the common case, the size of the Delaunay triangulation and the number of Voronoi vertices still tend to increase exponentially with the dimension.

Still, many efficient algorithms have been written for computing both Delaunay triangulations and Voronoi diagrams in $\mathbb{R}^d$. Some well known solutions include the Bowyer-Watson algorithm [13], [14], the widely used Quickhull [15], and a graph based approach [16] that attempts to mitigate storage requirements at the expense of computation time. It should be noted that although these algorithms are extremely efficient, none of them are believed to scale past eight dimensions for large data sets, due to the exponential nature of the problem.

To avoid the curse of dimensionality, this paper considers the case described in Section 2.2, where only a small number of Voronoi cells or umbrella neighbourhoods are required for the particular application. To save time, one could compute only the necessary subset, which may be considerably smaller than the complete diagram or triangulation. A similar approach has been taken before in the C++ library `Voro++` [17], which computes Voronoi cells in three dimensions. However, the methodology in [17] is fundamentally different than the algorithm proposed in this paper, and was not extended to the arbitrary-dimensional case in its implementation.

3. Necessary Operations

In this section, the machinery for computing the umbrella neighbourhood of a vertex $\hat{p}$ in the Delaunay triangulation is developed. Recall that each Delaunay simplex corresponds to exactly one Voronoi vertex. Therefore, the Voronoi cell of the $\hat{p}$ will follow trivially. There are two operations that need to be defined and one data structure. The two operations are the construction of an initial Delaunay simplex and the completion of an open Delaunay facet. The data structure needed is the active facet list (AFL). These operations and data structures are also established as components in the construction of the complete Delaunay triangulations via *incremental construction* and *divide-and-conquer* paradigms in [18].

3.1. Growing a Delaunay Simplex

To begin, the following definition of a Delaunay face is useful.

Definition 4. *Let P be a set of points in $\mathbb{R}^d$. Let F be a k-face with vertices in P where $0 \leq k \leq d$. Then F is a Delaunay face if the smallest (radius) $(d-1)$-sphere $C(v, r)$ circumscribing F satisfies $B(v, r) \cap P = \emptyset$.*

Note that if $k = d$, then by Observation 1, F is a Delaunay simplex. The following lemma shows how to grow a complete Delaunay simplex from an arbitrary Delaunay face.

Lemma 1. *Let P be a set of points in $\mathbb{R}^d$ in general position, and let F be a Delaunay k-face with vertices $\phi \subset P$ where $k < d$. Let $\phi^* = \phi \cup \{p^*\}$ where $p^* \in P \setminus \phi$ minimizes the radius of the smallest $(d-1)$-sphere $C_{\phi \cup \{p\}}$ through the points in $\phi \cup \{p\}$, over all $p \in P \setminus \phi$. Then F^*, the $(k+1)$-face with vertices ϕ^*, is also a Delaunay face.*

The proof of this lemma has been omitted, but follows from a continuity argument. The key is to gradually deform the initial sphere C_0 that passes through ϕ by moving its center toward the projection of the minimizing point p^* on the linear manifold M of points equidistant from ϕ, and maintaining that the deformed sphere C^* must still pass through ϕ. Note that the radius of C^* increases monotonely, therefore by continuity, no point can "pass through" C^* (and

into its open ball B^*) before p^* is encountered, since p^* was chosen to minimize the radius of C^*.

The smallest $(d-1)$-sphere containing a single point is simply the point itself, whose open ball is the empty set. Therefore, any point in P is a Delaunay 0-face trivially. So, starting with an arbitrary point in P and applying the following algorithm d times, the vertices of a full Delaunay simplex S are found. The correctness of this approach follows immediately from Lemma 1.

Algorithm 1, computes the vertices $\phi^* \supset \phi$ of a Delaunay $(k+1)$-face from the vertices ϕ of a Delaunay k-face.

Let P be a set of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$ in general position.
Let $\phi \subset P$ be the vertices of a Delaunay k-face where $k < d$.
Let r_i denote the minimum radius of a $(d-1)$-sphere containing ϕ and the point p_i.

$r_{min} := \infty$;
for $i := 1, \ldots, n$ **do**
  if $r_i < r_{min}$ **and** $p_i \notin \phi$ **then**
    $p^* := p_i$;
    $r_{min} := r_i$;
  end if
end for
return $\phi^* := \phi \cup \{p^*\}$;

3.2. Completing an Open Facet

Before presenting the algorithm for completing an open facet, the following observation is helpful. A visual representation of Observation 2 is presented in Figure 3.

Observation 2. *Let $DT(P)$ be the Delaunay triangulation of a set of points P in $\mathbb{R}^d$. Given a facet F of some simplex $S \in DT(P)$, let $p_1, \ldots, p_n$ be a sequence of points in P and in a halfspace H with hyperplane boundary containing F. Define the circumspheres $C_1, \ldots, C_n$ with corresponding open balls $B_1, \ldots, B_n$ such that each C_k contains σ and p_k. Assume the sequence $p_1, \ldots, p_n$ satisfies $p_k \in B_{k+1}$ for all $1 \leq k < n$. Then $B_1 \cap H \subset B_2 \cap H \subset \cdots \subset B_n \cap H$.*

The proof of this observation has been omitted but follows from another continuity argument, similar to that of Lemma 1. Given Observation 2, the following algorithm is proposed for completing an open facet of a Delaunay simplex.

Algorithm 2, computes the set of vertices s of a Delaunay simplex in the halfspace H from the vertices σ of a Delaunay facet F lying in the boundary hyperplane of H.

Let P be a set of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$ in general position.
Let σ be the vertices of a facet F of a Delaunay simplex.
Let H be a halfspace with hyperplane boundary containing F.

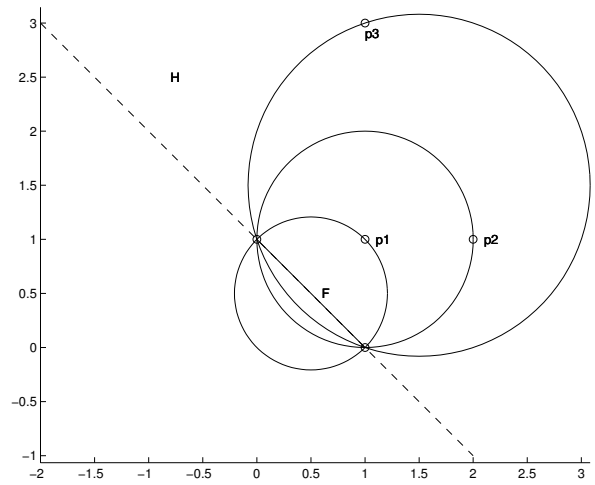

Figure 3. A 2D visualization of Observation 2. The solid line segment is a facet F, H is the halfspace above the dashed line, and $\{p1, p2, p3\}$ is a sequence satisfying $p1 \in B_2$ and $p2 \in B_3$.

Let $P_H = (P \setminus \sigma) \cap H$.
Let B_i denote the open ball defined by the $(d-1)$-sphere containing σ and p_i.

$B := H$;
$p^* := p_0$; Note, p_0 is a dummy vertex.
for $i := 1, \ldots, n$ **do**
  if $p_i \in P_H$ **and** $p_i \in B$ **then**
    $p^* := p_i$;
    $B := B_i$;
  end if
end for
if $p^* = p_0$ **then**
  σ must define a facet of $CH(P)$
  return $\emptyset$;
else
  return $s := \sigma \cup \{p^*\}$;
end if

The correctness of Algorithm 2 follows from Observation 2, which guarantees that the final point p^* must satisfy $p^* \in B_i$ for all $p_i \neq p^*$.

3.3. The Active Face List

During the construction of the Delaunay umbrella neighbourhood, it is often necessary to track which facets of the working set of Delaunay simplices are "open" and which are "closed." Note that every facet of a simplex in any triangulation that is *not* a facet of the convex hull is shared between exactly two simplices in that triangulation. Given a facet F, if only one simplex containing F has been constructed thus far, then F is said to be "open" since it still needs to be matched to a second simplex. If F is shared between two simplices already, then F is said to be "closed" since no other legal simplices can contain F.

The AFL is a data structure for storing the list of open facets. The key idea is that whenever a new simplex S is

constructed, any facet of S that contains $\hat{p}$ as a vertex either closes a previously computed open facet from the umbrella neighbourhood, or creates a new open facet that must be closed in the future. Therefore, whenever the algorithm constructs a new simplex S, any facet F of S that contains $\hat{p}$ should be used to update the AFL. If F is already in the AFL, then the existing instance of F must be deleted from the AFL since F has been closed by the new simplex S. If F is not in the AFL, then it must be added since it is an open facet. The update function is therefore outlined as follows.

Algorithm 3, updates the AFL with a facet F of a Delaunay simplex.

inputs: AFL A, Facet F
if $F \in A$ **then**
  $A := A \setminus \{F\}$;
else
  $A := A \cup \{F\}$;
end if
return

Note that in higher dimensions, the AFL could grow exponentially large. Therefore, a more sophisticated structure than a static array is required for its implementation. In this work, a dynamic array object that mimics the C++ `Vector` class is used. In particular, the AFL is initialized to an array of length eight, then its length is doubled whenever the size of the AFL overflows the current allocation. Note that for optimal performance, a hash table structure has been previously recommended to improve AFL query speed [18].

4. Algorithm Description and Analysis

4.1. Algorithm Description

In this section the computation of the Delaunay umbrella neighbourhood of $\hat{p} \in P \subset \mathbb{R}^d$, will be detailed. Using the two operations outlined in Section 3 and the AFL structure, it is possible to grow an initial Delaunay simplex from $\hat{p}$ then "fan out" to complete the entire umbrella neighbourhood of $\hat{p}$. In each iteration, a Delaunay simplex S incident to vertex $\hat{p}$ is created and every facet of S that contains $\hat{p}$ as a vertex must be used to "update" the AFL.

Algorithm 4 shows how to compute the Delaunay umbrella neighbourhood of a point $\hat{p}$. Alternatively, instead of outputting the vertices of all the simplices in the umbrella neighbourhood of $\hat{p}$, one could compute the circumcenters of these simplices to obtain the Voronoi cell containing $\hat{p}$.

Algorithm 4, computes the Delaunay umbrella neighbourhood of some $\hat{p} \in P$.

Let P be a set of n points in $\mathbb{R}^d$ in general position.
Let $\hat{p} \in P$.
Let A be an AFL.
Let MakeFirstSimplex(P, $\hat{p}$) be a function that grows the set of vertices of a Delaunay simplex starting from $\hat{p}$, as described in Algorithm 1.
Let CompleteSimplex(σ, P) be a function that completes the open facet defined by the set of vertices σ with a point from P that is on the open side of the hyperplane containing σ; as described in Algorithm 2.
Let Update(σ, A) be a function that updates A using the facet with vertices σ, as described in Algorithm 3.
Let DeleteFacet(σ, A) be a function that deletes the facet with vertices σ from A.
Let GetFacet(A) be a function that returns the vertices σ of some facet in A.

s := MakeFirstSimplex(P, $\hat{p}$);
$N := \{s\}$;
$A := \emptyset$;
for $s_i \in s$ **do**
  if $s_i \neq \hat{p}$ **then**
    $\sigma := s \setminus \{s_i\}$;
    Update(σ, A);
  end if
end for

while $A \neq \emptyset$ **do**
  σ := GetFacet(A);
  s := CompleteSimplex(σ, P);
  if $s = \emptyset$ **then**
    The facet with vertices σ was a facet of $CH(P)$
    DeleteFacet(σ, A);
  else
    for $s_i \in s$ **do**
      if $s_i \neq \hat{p}$ **then**
        $\sigma := s \setminus \{s_i\}$;
        Update(σ, A);
      end if
    end for
    $N := N \cup \{s\}$;
  end if
end while
return N;

The correctness of Algorithm 4 follows from the correctness of the operations in Sections 3.1 – 3.3. The simplices constructed will always be Delaunay since Algorithms 1 and 2 are correct. Furthermore, since every open facet containing $\hat{p}$ as a vertex is "updated" in the AFL, the algorithm will terminate only once *every* facet has been closed (or is a facet of $CH(P)$). This ensures that all simplices incident to $\hat{p}$ are found, and since only finitely many simplices are in the umbrella neighbourhood of $\hat{p}$, the algorithm terminates in finite time.

4.2. Time Complexity

The construction of the first simplex, as defined in Algorithm 1, can be formulated as a sequence of least squares (LS) problems ranging in size from $2 \times d$ to $d \times d$. Each LS problem can be solved in $\mathcal{O}(d^3)$ time. At all $d-1$ sizes, one must solve up to n LS problems, taking the point that produces the minimum residual as the next vertex.

Therefore, the total computation time for the first simplex will be $\mathcal{O}(nd^4)$.

To complete a facet requires at most n linear solves, performed in $\mathcal{O}(nd^3)$ total time. Therefore, the total time complexity for constructing all simplices after the initial simplex is given by $\mathcal{O}(nd^3\kappa)$, where κ is the total number of simplices in the Delaunay umbrella neighbourhood (or equivalently, the number of Voronoi vertices). This makes the total time complexity $\mathcal{O}(nd^4+nd^3\kappa)$. In general, κ could be much greater than d, so this can be further reduced to $\mathcal{O}(nd^3\kappa)$. As seen in Section 6 (Table 2), for uniformly distributed data, κ tends to scale with d independent of n, making the total time complexity approximately linear in n.

4.3. Space Complexity

Recall from Section 2.3 that the size of the complete Delaunay triangulation and Voronoi vertex list grows exponentially with the dimension. Therefore, space complexity is equally as concerning as time complexity since, for large d, one cannot store the exponentially sized triangulation in memory. Using Algorithm 4, the space required for computing the umbrella neighbourhood is reduced to:

- The space required to store the entire umbrella neighbourhood/Voronoi vertex list, which is proportional to the number of simplices in the umbrella neighbourhood and vertices of the Voronoi cell.
- The space required for the AFL, which is determined by the maximum number of open facets.

As will be seen in Section 6 (Table 3), the maximum space requirement of the AFL is significantly less than that of the umbrella neighbourhood itself. Therefore, the space requirement of this algorithm is limited only by the size of the output (i.e., the number of simplices in the umbrella neighbourhood), making this a space efficient technique.

5. Handling Degeneracy

5.1. The Challenges of Degeneracy

An important assumption that has been made up until this point is that P is in general position. However, in real world applications, it is possible that P could be some degenerate point set (for example, consider a grid aligned data set). There are two types of degeneracies to consider.

- P could be contained in some lower-dimensional linear manifold.
- There could exist $d+2$ or more points in P that lie on the same $(d-1)$-sphere.

The situation where all points lie in a lower-dimensional linear manifold will always result in a situation where no new point can be "added" to the set of vertices during the construction of the first Delaunay simplex without making some face degenerate. This situation is easily detected via a check for rank deficiency and need not be handled since users can apply dimension reduction techniques to construct an equivalent non degenerate problem.

In the case where more than $d+1$ points lie on some $(d-1)$-sphere, one would like to still obtain the umbrella neighbourhood of $\hat{p}$ in a Delaunay triangulation, though it will no longer be unique. However, the presence of the degeneracy causes Algorithm 4 to break down, as it violates the assumption that P is in general position.

To understand what could go wrong in Algorithm 4 under this kind of degeneracy, consider the two-dimensional umbrella neighbourhood shown in Figure 4. Because there is no unique triangulation, it is possible to identify conflicting simplices, which are both members of a Delaunay triangulation but cannot be members of the same triangulation.

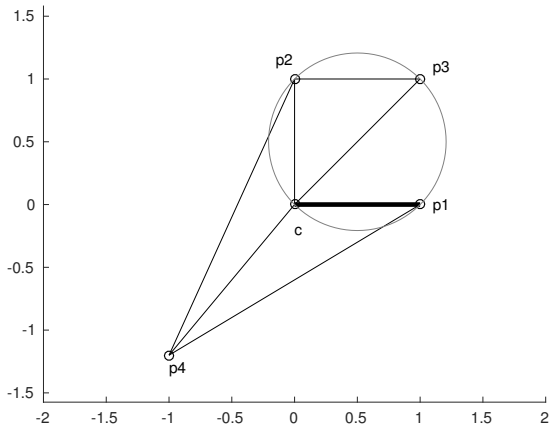

Figure 4. Consider this two-dimensional example, where the umbrella neighbourhood of the central point c is being computed. Algorithm 4 has been used to compute all the simplices shown, and in the current step it has chosen the bold edge between c and $p1$ from the active facet list. If $p3$ is matched with this open facet, then the other open facet between c and $p3$ will be closed and the algorithm terminates correctly. However, note that due to the degeneracy (points c, $p1$, $p2$, and $p3$ are cocircular), $p2$ is also a valid option. If $p2$ is chosen instead of $p3$, then the algorithm could continue in an infinite loop, never properly matching all open facets.

Note that this failure to properly "match" facets is not an issue in the case of an acyclic walk. However, properly closing open facets is a key condition for the termination of Algorithm 4. Therefore, in a fundamental way, Algorithm 4 is incompatible with degenerate data.

5.2. Dealing with Degeneracies

Since degeneracies occur with zero probability, a solution to the problems caused by degeneracy is to perturb each $p \in P$ by some small random vector of magnitude ϵ such that the perturbed set $\rho(P) = P + Q_\epsilon$ is in general position [19]. Note Algorithm 4 will succeed on the new point set $\rho(P)$, and the computed triangulation $DT(\rho(P))$ will be within a perturbation of magnitude ϵ of the true triangulation $DT(P)$. Therefore, the computed triangulation $DT(\rho(P))$ could be used with the original point set P as a backward stable estimate for $DT(P)$.

The magnitude ϵ of the perturbations is important here. If ϵ is chosen too large, the computed triangulation $DT(\rho(P))$ will not reflect the true triangulation $DT(P)$. However, if ϵ is chosen too small relative to the machine precision and problem scale, then the effect of round off errors could recreate the degeneracies. To simplify the issues, the point set P should first be translated so that $\hat{p}$ is centered at the origin, then P should be scaled so that its diameter is 1. Now, the value ϵ can be chosen strictly in relation to the unit roundoff without accommodating scale. In practice, a good choice for ϵ would be $\epsilon = \sqrt{\mu}$ where μ is the unit roundoff.

6. Empirical Results

A serial implementation of the proposed algorithm has been coded in ISO Fortran 2003. This code was tested for correctness against the standard implementation of Quickhull on pseudo random data sets ranging in dimension from $d = 2$ to $d = 5$ and ranging in size from $n = 20$ to $n = 32,000$. Note that due to the exponential nature of the problem, Quickhull becomes prohibitively slow for large data sets in high-dimensional space. Where feasible, the correctness was confirmed by ensuring that all computed simplices were also members of the triangulation computed by Quickhull. The following run times were gathered on an Intel i7-3770 CPU @3.40 GHz running CentOS release 7.3.1611.

To gather the data seen in Tables 1, 2, and 3, the umbrella neighbourhood was computed for the central point in uniformly distributed point sets ranging in size from 2K to 32K and ranging in dimension from two to five. At each size and dimension combination, 20 independent trials were performed over different point sets. Table 1 gives the average observed runtimes at each size and dimension, Table 2 gives the average number of simplices in the Delaunay umbrella neighbourhood at each size and dimension, and Table 3 gives the average maximum size attained by the AFL at each size and dimension. All input data sets consist of pseudo randomly generated points in the unit hypercube, generated using the Fortran intrinsic random number generator. Times were recorded with the Fortran intrinsic `CPU_TIME` function, which is accurate up to either microsecond resolution or the precision of the system clock.

Table 1. AVERAGE RUNTIME IN SECONDS FOR COMPUTING THE UMBRELLA NEIGHBOURHOOD OF THE CENTRAL POINT FOR n PSEUDO-RANDOMLY GENERATED INPUT POINTS IN d-DIMENSIONAL SPACE.

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
$d = 2$	0.1 s	1.6 s	6.3 s	25.0 s
$d = 3$	0.1 s	1.8 s	7.0 s	27.8 s
$d = 4$	0.2 s	2.0 s	7.6 s	30.0 s
$d = 5$	0.3 s	2.6 s	9.2 s	34.1 s

As seen, the proposed algorithm performs well, computing the umbrella neighbourhood relatively quickly. Note that the maximum space requirement of the AFL is significantly less than that of the umbrella neighbourhood, making this a space efficient solution. For comparison, consider the tables in Section 5 of [16], which give statistics for the computation of the complete Delaunay triangulation via state-of-the-art methods.

Table 2. AVERAGE NUMBER OF SIMPLICES IN THE UMBRELLA NEIGHBOURHOOD OF THE CENTRAL POINT FOR n PSEUDO-RANDOMLY GENERATED INPUT POINTS IN d-DIMENSIONAL SPACE.

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
$d = 2$	6.30	5.90	6.25	6.65
$d = 3$	28.10	28.90	29.50	29.10
$d = 4$	151.90	170.55	173.60	161.00
$d = 5$	1122.90	1115.70	1111.00	1038.70

Table 3. AVERAGE MAX SIZE OF THE AFL WHILE COMPUTING THE UMBRELLA NEIGHBOURHOOD OF THE CENTRAL POINT FOR n PSEUDO-RANDOMLY GENERATED INPUT POINTS IN d-DIMENSIONAL SPACE.

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
$d = 2$	2	2	2	2
$d = 3$	11.85	12.30	12.55	12.15
$d = 4$	75.30	82.50	83.00	82.70
$d = 5$	658.05	643.05	642.55	610.65

7. Conclusion and Future Work

In this paper, a new algorithm is proposed for computing the Delaunay umbrella neighbourhood of a point, or equivalently, a single Voronoi cell. The proposed algorithm is empirically shown to scale efficiently. In future work, the performance of the algorithm could be improved by implementing the AFL as a hash table as recommended in [18]. Also, it would be interesting to compare its performance with that of the Voro++ algorithm described in [17].

References

[1] S. Fortune, "A sweepline algorithm for voronoi diagrams," *Algorithmica*, vol. 2, no. 1-4, p. 153, 1987.

[2] P. Su and R. L. S. Drysdale, "A comparison of sequential delaunay triangulation algorithms," in *Proceedings of the eleventh annual symposium on Computational geometry*. ACM, 1995, pp. 61–70.

[3] V. Klee, "On the complexity of d-dimensional voronoi diagrams," *Archiv der Mathematik*, vol. 34, no. 1, pp. 75–80, 1980.

[4] M. de Berg, O. Cheong, M. Van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer-Verlag Berlin Heidelberg, 2008.

[5] R. Sibson, "A vector identity for the dirichlet tessellation," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 87, no. 1. Cambridge University Press, 1980, pp. 151–155.

[6] L. Traversoni, "Natural neighbour finite elements," *WIT Transactions on Ecology and the Environment*, vol. 8, 1970.

[7] J. W. Brandt and V. R. Algazi, "Continuous skeleton computation by voronoi diagram," *CVGIP: Image understanding*, vol. 55, no. 3, pp. 329–338, 1992.

[8] S.-W. Cheng, T. K. Dey, and J. Shewchuk, *Delaunay Mesh Generation*. CRC Press, 2012.

[9] V. Rajan, "Optimality of the delaunay triangulation in $\mathbb{R}^d$," *Discrete & Computational Geometry*, vol. 12, no. 2, pp. 189–202, 1994.

[10] H. Edelsbrunner and E. P. Mücke, "Three-dimensional alpha shapes," *ACM Transactions on Graphics (TOG)*, vol. 13, no. 1, pp. 43–72, 1994.

[11] C. H. Rycroft, G. S. Grest, J. W. Landry, and M. Z. Bazant, "Analysis of granular flow in a pebble-bed nuclear reactor," *Physical review E*, vol. 74, no. 2, p. 021306, 2006.

[12] S. Deshpande, L. T. Watson, and R. A. Canfield, "Multiobjective optimization using an adaptive weighting scheme," *Optimization Methods and Software*, vol. 31, no. 1, pp. 110–133, 2016.

[13] A. Bowyer, "Computing dirichlet tessellations," *The Computer Journal*, vol. 24, no. 2, pp. 162–166, 1981.

[14] D. F. Watson, "Computing the n-dimensional delaunay tessellation with application to voronoi polytopes," *The Computer Journal*, vol. 24, no. 2, pp. 167–172, 1981.

[15] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, pp. 469–483, 1996.

[16] J.-D. Boissonnat, O. Devillers, and S. Hornus, "Incremental construction of the delaunay triangulation and the delaunay graph in medium dimension," in *Proceedings of the twenty-fifth annual symposium on Computational geometry*. ACM, 2009, pp. 208–216.

[17] C. H. Rycroft, "Voro++," Lawrence Berkeley National Laboratory, Tech. Rep., 2009.

[18] P. Cignoni, C. Montani, and R. Scopigno, "Dewall: A fast divide & conquer delaunay triangulation algorithm in $\mathbb{E}^d$," *Computer-Aided Design*, vol. 30, no. 5, pp. 333–341, 1998.

[19] H. Edelsbrunner and E. P. Mücke, "Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms," *ACM Transactions on Graphics (TOG)*, vol. 9, no. 1, pp. 66–104, 1990.