A parallel algorithm for computing Voronoi diagram of a set of circles using touching disc and topology matching

Submission number: 34

Abstract

We present a parallel algorithm for computing the Voronoi diagram of a set of circles, C. The algorithm starts with a parallel computation of the Voronoi cell of each circle $c_i \in C$, using a subset of circles, $C_i \subset C$, and a reduced touching disc method. Unlike the actual touching disc method, a reduced touching disc method does not compute antipodal discs (ADs) between all pairs of circles. We employ the Delaunay graph, $\mathbb{D}$, of the center points of circles in C to set an initial C_i. For computing an initial Voronoi cell for $c_i \in C$, it is enough to compute ADs between c_i and each of the circles $c_j \in C_i$. Computation of Voronoi cells updates initially assumed C_i, and the algorithm iteratively computes those Voronoi cells until no further updating of C_i occurs. A parallel procedure corrects the Voronoi cells, iteratively, using a topology matching approach. A reduced touching disc method and a cell-wise computation of the Voronoi diagram allow input in non-general position. Results and performance of the algorithm show robustness and speed of the algorithm in handling a large set of circles.

Keywords: Voronoi diagram of circles, Touching disc, Topology matching, Delaunay graph, Antipodal disc

1. Introduction

A Voronoi diagram, which is constructed for a set of geometric objects, divides the space into regions called Voronoi cells such that there is a Voronoi cell for each object in the set. Every point in the Voronoi cell is nearer to the corresponding object than any other object in the set, and the boundary of the Voronoi cells is a set of points nearest to more than one object. Applications of the Voronoi diagram can be seen in many fields of science and engineering Aurenhammer (1991). Image segmentation, computer-assisted diagnostics, disease research, dimensional reduction, 3D printing, motion planning and collision avoidance are a few examples to cite Cornea et al. (2007); Wang et al. (2019); Zivanic et al. (2012); Zheng et al. (2021). Voronoi diagram of a set of circles of non-uniform radii is different from that of a set of points in both topological and geometric aspects. Voronoi diagram of circles, also called additively weighted Voronoi diagram, finds application in bundling of electric-wires Sugihara et al. (2004); Ryu et al. (2020), sensor deployment Mahboubi and Aghdam (2013), visualization of particle growth Anton et al. (2009) etc.

Lee and Drysdale (1981) proposed an algorithm, which is considered as an initial work for computing the Voronoi diagram of a set of non-intersecting circles, with a complexity of $O(n\,log^2 n)$, where n is the number of circles in the input set. Algorithm by D.-S. Kim *et al.*, use the Voronoi diagram of center points of circles to arrive at an $O(n^2)$ algorithm for computing the Voronoi diagram of circles that can be intersecting and of varying radii Kim et al. (2001a,b). Sugihara (1993) also used the Voronoi diagram of points to get an approximated Voronoi diagram of circles. Here, the set of points are sampled from the boundary of the input circles. Even though robust and $O(n\,log\,n)$ algorithms are available for computing the Voronoi diagram of point set, Sugihara's algorithm requires a large number of points to get a reasonable accuracy in topology and geometry. Jin et al. (2006) proposed a sweep line algorithm that computes the Voronoi diagram of non-uniform and intersecting circles with a $O(n^2\,log\,n)$ complexity; for set of non-intersecting circles, the complexity reduces to $O(n\,log\,n)$. Algorithm of Lee et al. (2016) follows a topology-oriented incremental approach and gives a linear performance even though the complexity is of $O(n^2)$. A touching disc method by Sundar et al. (2020) for computing the Voronoi diagram of a set of planar closed curves can be used

to find the Voronoi diagram of a set of non-intersecting circles with $O(n^2)$ time complexity. Mukundan and Muthuganapathy (2021) have proposed a dynamic sampling approach for the exact computation of the Voronoi diagram of a set of circles, permitting intersection of circles with $O(n\,log\,n)$ complexity.

The Voronoi diagram of a set of circles consist of Voronoi edges (set of points nearest to two circles) and Voronoi vertices (points nearest to more than two circles). During the computation of the Voronoi diagram of circles, the algorithm may come across Voronoi vertices nearest to more than three circles in the input set. An input set with such degeneracy is called input set in non-general position. There are some preconditioning techniques like symbolic perturbation of input set for dealing with such degeneracies in the input set Edelsbrunner and Mücke (1990); Yap (1990); Sugihara (1992). Devillers et al. (2015) use a sequence of perturbation parameters to perturb the input set. Emiris and Karavelas (2006) use the analysis of predicates to exactly compute the additively weighted Voronoi diagram in the plane. Sugihara and Iri (1992) compute the Voronoi diagram of a set of points and Lee et al. (2016) compute the Voronoi diagram of a set of circles following a topology-oriented approach to address the issue of degeneracies in the input set. Mukundan and Muthuganapathy (2021) used a modified touching disc method to address the input set in non-general position.

Many algorithms developed over the years for computing the Voronoi diagram of circles have addressed computational complexities and robustness to a great extent. However, instructions or computations followed in most existing algorithms are sequential and difficult to implement parallel programming. As the CPU and GPU based parallel computation significantly reduce the computational time, algorithms that can be converted into a parallel program are getting more relevant nowadays. Parallel programming demands algorithms that can be converted into subprograms that can be executed in parallel. An initial work in this regard is by Aggarwal et al. (1988), where they proposed a parallel algorithm for computing the Voronoi diagram of a set of points. The parallel algorithm of Goodrich et al. (1993) computes the Voronoi diagram, where the objects involved are either lines or points. A GPU based jump flooding algorithm (JFA) is used to compute: a 2D Voronoi diagram of a point set by Rong and Tan (2006) and an evenly-spaced tessellation on a surface by Rong et al. (2010). Yuan et al. (2011) used an improved JFA to compute a 2D generalized Voronoi diagram on GPU. A GPU accelerated algorithm of Cao et al. (2014) uses a two-stage iterative algorithm where the first stage is parallel, whereas the corrective step in the second stage is sequential. Peterka et al. (2014) proposed a high-performance parallel algorithm for computing the 3D Voronoi diagram of a point set with a distributed memory. González (2016) also proposed a memory distributed parallel algorithm for a 3D point set that subdivides the bounding box for local computation of the Voronoi diagram. Ray et al. (2018) proposed a k-NN based parallel algorithm that performs best for computing the 3D Voronoi diagram for evenly distributed point set or point set of smoothly varying density. Bernaschi et al. (2017) employ a GPU based parallel implementation for updating the topological changes in a dynamic Voronoi diagram where the generating points are moving. Liu et al. (2020) proposed a parallel algorithm that computes the Voronoi diagram of a set of points in a restricted domain. The algorithm by Hu et al. (2017); Li et al. (2019) computes the Voronoi vertices and edge topology for a set of spheres using the lower envelope of bisectors. The same algorithm could be implemented for computing the Voronoi diagram of a set of circles. To the best of our knowledge, there is no other algorithm for computing the Voronoi diagram of a set of circles using parallel computing. Computations in their algorithm may be done independently for each Voronoi cell. However, that would involve finding bisector with each of the other circles leading to a complexity of $O(n^2)$.

This paper proposes a parallel algorithm to compute the Voronoi diagram of a set of circles. The main challenge in the parallel computation of the Voronoi diagram is to divide the input set into subgroups and make a local computation of the Voronoi diagram restrained in each sub-group. Many subdivision algorithms assume the input set is evenly-spaced or input of smoothly varying density. In our algorithm, the number of subdivisions is always equal to the size of the input set (more importantly, we are not doing any subdivision of the bounding box to achieve it). In other words, we compute the Voronoi cell of each circle independently using a small subset of input circles such that each Voronoi cell computation is local to that subset. This facilitates distributed memory allocation for each sub-program. We use the Delaunay graph of the center point of circles to define the subset of circles assigned to each circle for computing its Voronoi cell. This, in turn, helps to achieve a linear memory requirement for computing the Voronoi diagram. Most parallel

algorithms are iterative and require some corrective steps to converge. Our algorithm computes Voronoi cells using a two-stage iteration procedure. The first stage of iteration uses a reduced touching disc method there by avoiding the need to compute bisectors (a computationally expensive and numerically intractable one) at the start. The result obtained from the first stage is corrected using a second stage iteration that follows a topology matching procedure. We also demonstrate that the computations are much faster for large input sets even using only the CPU. Following are the highlights of the algorithm.

- Voronoi diagram computation for a set of circles of size n is divided into n sub-programs with a distributed memory for each sub-program.
- Voronoi cells are computed in parallel, using a small subset of input circles.
- identifying subsets using the Delaunay graph of center points of circles makes the total memory requirement linear in n.
- handles input in non-general position without any special conditioning of the input set.
- a topology matching with a touching disc method ensures topological correctness and geometric accuracy of the computed Voronoi diagram.
- offers a complexity of $O(n \log n)$ when implemented in sequential computing.

The remainder of this paper is organised as follows. Sections 2 and 3 explain the terminologies used in this paper and a brief description of the touching disc method, respectively. Section 4 discusses the basic idea of the algorithm. The reduced touching disc method and computation of initial Voronoi cells are then explained in Section 5. Section 6 presents the topology matching procedure, and Section 7 presents the algorithm and the parallel framework. In Section 8, the results of the algorithm are presented and discussed. Finally, Section 9 concludes the paper.

2. Preliminaries

Let $C = \{c_i | i = 1, 2, ...n\}$ represent a set of input circles in $\mathbb{R}^2$, where n is the size of C and $c_i = c_i(p_i, r_i)$ represents a circle with center at $p_i = p_i(x_i, y_i)$ and radius $r_i \geq 0$. $P = \{p_i | i = 1, 2, ...n\}$ represents set of center points of all input circles. Let $VD(C)$ be the Voronoi diagram, in $\mathbb{R}^2$, for the set of circles C, computed in Euclidean metric. The input circles are assumed to be non-intersecting. However, the proposed algorithm can also handle inputs containing intersecting circles and identifies circles that are completely contained in another circles.

Definition 1. ***Touching disc*** *is a disc that is tangential to a set of circles and doesn't contain any of them. The point at which the touching disc touches a circle is called* ***footpoint*** *(Figure 2(a)-(b)).*

Definition 2. ***Antipodal Disc (AD)*** *between two circles is the touching disc with minimum radius and having antipodal footpoints (i.e., they are diametrically opposite).*

AD between circles c_a and c_b is represented as AD_{ab} (brown disc in Figure 2(c)).

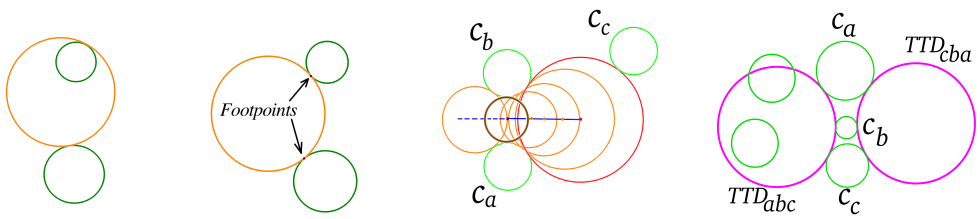
Definition 3. *A touching disc to three circles is called* ***Three Touch Disc (TTD).***

There can be two TTDs between c_a, c_b and c_c, represented as TTD_{abc} and TTD_{cba} (Figure 2(d)).

Definition 4. *A touching disc of a set of circles is said to be* ***empty*** *if it does not intersect or contain any of the other circles from C.*

TTD_{abc} in Figure 2(d) is not empty as it either contains/intersects at least one circle other than c_a, c_b and c_c. TTD_{cba} is empty.

Definition 5. *A* ***Voronoi disc*** *is a touching disc that is empty and has more than one footpoint.*



(a) (b) (c) (d)

(d) Two TTDs exist between c_a, c_b, and c_c. TTD_{abc} is not empty as it contains\intersects one input circle.

(d) Two TTDs exist between c_a, c_b, and c_c. TTD_{abc} is not empty as it contains\intersects one input circle.

Figure 1: Touching discs (in this paper, input circles are shown in green color, unless specified otherwise). (a) Not a touching disc as it contains a circle to which it is tangent. (b) A touching disc and footpoints. (c) All tangent circles between c_a and c_b are touching discs between them, and the AD is shown in brown. TTD between c_a, c_b and c_c is shown in red.
(d) Two TTDs exist between c_a, c_b, and c_c. TTD_{abc} is not empty as it contains\intersects one input circle.

Definition 6. *A **Voronoi edge** between two circles is the locus of the center of a Voronoi disc, which has footpoints with these input circles.*

Definition 7. *An empty TTD is called a **branch disc**, and its center is called a **branch point**.*

Definition 8. *The radius of the Voronoi disc varies along a Voronoi edge, and it is expressed as a function called **Voronoi radius function** $R(u)$, where u is a parameter representing a point on the Voronoi edge. Voronoi disc for which $R(u)$ reaches a local maximum is called **disc of** R_{max}, whereas that of local minimum is called **disc of** R_{min}.*

Definition 9. *A Voronoi edge or a portion of a Voronoi edge bounded by the discs of R_{max} and R_{min}, between which $R(u)$ varies monotonically, is called a **Voronoi segment**.*

In Figure 2(a), one of the Voronoi segments is highlighted in cyan color, and corresponding discs of R_{max} and R_{min} are shown in red color. As $VD(C)$ is the union of all Voronoi segments, every Voronoi segment can be identified in the touching disc method using the disc of R_{min} corresponding to each of the Voronoi segments.

2.1. Directed edges

Directed edge (DE) is a data structure used to track the computation of Voronoi segments. A DE corresponds to a Voronoi segment and is graphically represented as a straight line segment with an arrowhead such that no point of its Voronoi segment lies at the left side of the DE. The endpoints of a DE are the footpoints of the *disc of* R_{min}, of the Voronoi segment, with c_a and c_b. A DE holds information such as the circles between which the DE is defined, the direction of the DE, and the radius, footpoints and center of the *disc of* R_{min}.

An empty AD acts as the *disc of* R_{min} for two Voronoi segments (Please refer to Definition 9). These Voronoi segments are represented by the DEs (DE_{ab} and DE_{ba}, which are shown separately in Figures 2(c) and (d), along with their corresponding Voronoi segment). In Figure 2(b)-(e), the *disc of* R_{min} is an empty AD, whereas, in Figure 2(a), a Voronoi segment is highlighted in cyan color for which the *disc of* R_{min} is an empty TTD. It may be noted that *disc of* R_{max} is always a branch disc; for the Voronoi segment extending to infinity, it is assumed that the branch point is at infinity and the *disc of* R_{max} is of radius ∞. In Figures 2(c) and (d), the center of the *disc of* R_{max} of the Voronoi segment is at infinity and lies at the right side of the DE.

3. A brief description of touching disc method

In touching disc method, ADs between all pairs of input circles are computed. These ADs are sorted in the non-decreasing order of radii and inserted into a list $\mathbb{L}$. Each of these ADs is tested for emptiness in the

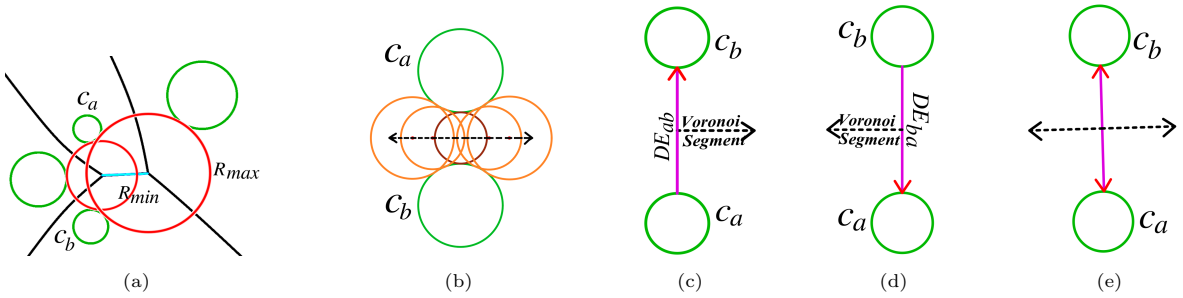


Figure 2: Voronoi segments and directed edges (DEs). (a) The Voronoi segment between c_a and c_b, bounded between the discs of R_{min} and R_{max}, is highlighted in cyan color. (b) AD_{ab} between c_a and c_b acts as the *disc of* R_{min} for two Voronoi segments (in dashed line). (c) Voronoi segment corresponding to DE_{ab}, and (d) Voronoi segment corresponding to DE_{ba}. (e) DE_{ab} and DE_{ba} are shown together.

order of its radius. Once found empty, DEs are placed to mark corresponding Voronoi segments.

Definition 10. *Two DEs, DE_{ab} and DE_{cd} are said to be consecutive if and only if either c_b and c_c or c_a and c_d represent the same circle .*

3.1. Parametric closeness of DEs

If we consider a circle c_a in the input set, there can be many DEs connected to c_a during the course of computation. Let $p(x(t), y(t))$ be a point on the circle c_a, where $t \in [0, 1]$ parameterize the circle. DEs which are parametrically close to the point p are those DEs that have one of their endpoints on c_a and closer to the point p along c_a in either clockwise or anticlockwise direction than the endpoint of any other DEs on c_a. So, at a particular instance of computation, there can be a maximum of two DEs parametrically close to the point p; one in the clockwise direction and the second in the anticlockwise direction. Parameter t is used to measure the parametric closeness of DEs.

Parametric closeness of two DEs can be checked if and only if they have footpoint on a common circle (Refer to Figure 3 for parametric closeness of DEs). When two consecutive DEs, DE_{ab} and DE_{bc}, are found parametrically close with respect to their endpoint on c_b, the possibility of constructing TTD_{abc} is checked. TTD_{abc} is possible if and only if the center of a TTD between c_a, c_b and c_c lies at the right side of both DE_{ab} and DE_{bc}.

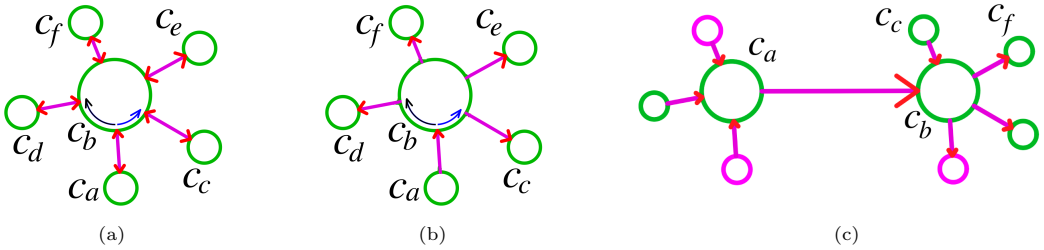
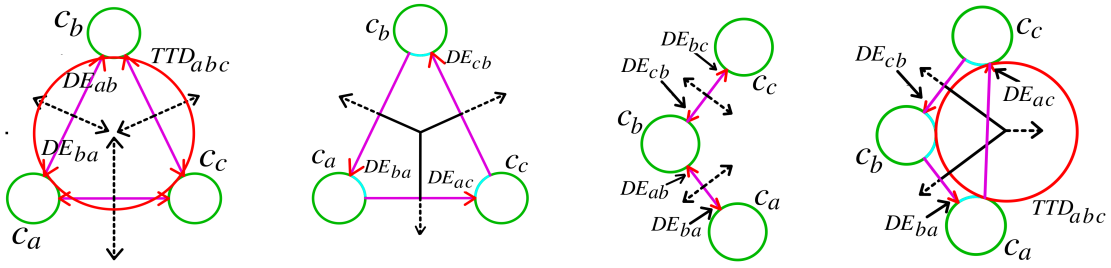


Figure 3: The parametric closeness of DEs (DEs are shown arbitrarily for the explanation). (a) DEs connected to c_b. DE_{bd} and DE_{db} are parametrically close to both DE_{ab} and DE_{ba} (clockwise, along c_b). DE_{bc} and DE_{cb} are also parametrically close to both DE_{ab} and DE_{ba} (anticlockwise, along c_b). (b) DE_{bd} and DE_{bc} are parametrically close and consecutive to DE_{ab}. (c) Circles (in pink) form TTDs with c_a and c_b on introducing DE_{ab}. TTD_{abc} will not be formed, as DE_{bc} is not present. TTD_{abf} will not be formed, as DE_{ab} and DE_{bf} are not parametrically close.

3.2. Processing TTDs and computing Voronoi segments

Parametric closeness restricts the number of TTDs to be formed for finding branch discs. When a TTD is formed, it is also inserted into the same list $\mathbb{L}$, maintaining the order of radii. When the turn of a TTD

(TTD_{abc}) arrives for processing, if all or at least two of its DEs (DE_{ab}, DE_{bc}, and DE_{ca} are DEs of TTD_{abc}) are present, it is empty, and TTD_{abc} will be identified as a branch disc (Figure 4(a)). Once a branch disc is identified, Voronoi segments of its DEs for which TTD_{abc} is the *disc of* R_{max} are computed. These DEs are deleted immediately after computing its Voronoi segments (Figure 4(b)). If one of the three DEs is absent (let it be DE_{ca}), it indicates that there is no Voronoi segment between c_a and c_c for which TTD_{abc} is the *disc of* R_{max}(Figure 4(c)). In this case, after computing Voronoi segments of DE_{ab} and DE_{bc}, DE_{ab} and DE_{bc} are deleted, and DE_{ac} is inserted to identify the Voronoi segment between c_a and c_c for which TTD_{abc} is the *disc of* R_{min}. Here, DE_{ac} is defined using the footpoints of TTD_{abc} on c_a and c_c (Figure 4(d)). The algorithm stops once all the discs in the list $\mathbb{L}$ are addressed.



(a) (b) (c) (d)

Figure 4: Processing TTDs. (a) Two DEs are defined using footpoints of each empty AD; in the figure, each arrowhead corresponds to a DE. (b) DE_{ab}, DE_{bc} and DE_{ca} are deleted after computing its Voronoi segments (shown in firm dark line). (c) DEs are not placed using the footpoint of AD_{ac}, as it is not empty. (d) DE_{ab} and DE_{bc} are deleted. DE_{ac} is introduced using the footpoints of TTD_{abc} (in red), and the corresponding Voronoi segment is also shown (with a dashed line labeled as VS_{ac}). In (a) and (b), the portion of circles used for computing Voronoi segments is highlighted in cyan color.

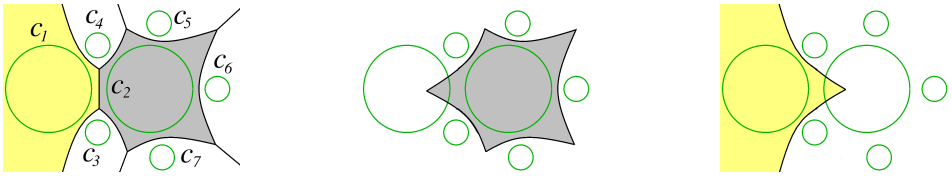
4. Basic idea of the algorithm

A 2D Voronoi diagram divides the plane into Voronoi cells, which are disjoint sets of points (as every point in a Voronoi cell is nearest to the corresponding circle than any other circle in the input set). If HP_{ij} is the set of all points nearest to c_i than to c_j, we can define a Voronoi cell, $V_i = \bigcap_{i \neq j} HP_{ij}$. This way, Voronoi cell of c_i is the lower envelope of bisectors between c_i and the remaining circles in the input (the nearest bisectors form the boundary or envelope of the Voronoi cell). The Voronoi diagram of a set of circles is shown in Figure 5(a), where the Voronoi cell of c_1 (in yellow) and that of c_2 (in grey) are highlighted. Neighbors of c_i are those circles that share Voronoi edge with c_i. If we know *a priori*, the set of circles that are neighbors to c_i, (instead of finding bisectors between c_i and each of the remaining circles in the input set), it is enough to construct the bisectors between c_i and circles in the neighbors to find the lower envelope. If the neighbors are correct, the computed Voronoi cells will be disjoint (Figure 5(b)-(c)). However, computation of bisector intersection and trimming of bisectors are required in the computation of Voronoi cell following a lower envelope method.

The touching disc method of Sundar et al. (2020) computes the Voronoi diagram by computing the portion of bisectors between circles that exactly contributes to the Voronoi diagram without any need of further trimming. Major disadvantages of the touching disc method are to compute AD between all pairs of circles (leading to $O(n^2)$ ADs) and non-feasibility to employ parallel computing as the algorithm is sequential. A modified form of touching disc method is used in this algorithm which computes the Voronoi cell of a circle using only a small subset of circles that are being assumed as its neighbors. The computed Voronoi cells are used to correct the initially assumed set of neighbors. The Voronoi cell of a circle for which the neighbor set got updated is computed again. This process repeats until no update occurs for the neighbors of the circles. This completes the first stage of iteration and provides a set of initial Voronoi cells, and it is not guaranteed that these Voronoi cells are correct. The Voronoi cell computed in the first stage of

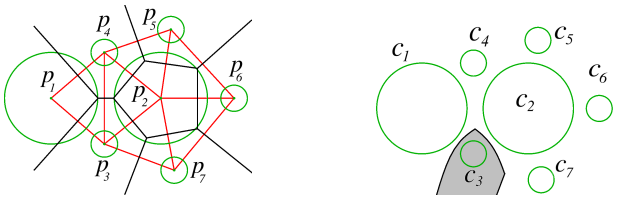
iteration is corrected by the second stage of iteration using a topology matching approach, considering the fact that the correct Voronoi cells will be disjoint.

The Delaunay graph $\mathbb{D}$, of the center points of circles, is considered for assigning an initial set of neighbors C_i to a circle $c_i \in C$. The Delaunay graph is a dual of the Voronoi diagram where a Delaunay edge connects two points if they share a Voronoi edge. If p_i is the center point of circle c_i, neighbors of p_2 in Figure 6(a) are p_3, p_4, p_5, p_6 and p_7. Hence, $C_2 = \{c_3, c_4, c_5, c_6, c_7\}$.



(a) (b) (c)

Figure 5: Voronoi cells are disjoint. (a) Voronoi cell of c_1 (in yellow) and that of c_2 (in grey). (b) Voronoi cell of c_2, where c_1 is not considered as its neighbor. (c) Voronoi cell of c_1, where c_2 is not considered as its neighbor.



(a) (b)

Figure 6: Delaunay graph of center points of circles are used to get an initial set of neighbors. (a) The Voronoi diagram (in black) and the Delaunay graph (in red) of the centers of circles. (b) The Voronoi cell of c_3 (in grey), computed using $C_3 = \{c_1, c_2, c_4, c_7\}$ shows c_1 and c_2 are neighbors. Hence, C_1 can be updated by adding c_2 and C_2 by adding c_1.

4.1. Updating initial set of circles used for computing a Voronoi cell

Figure 6(a) shows the center of c_2 is not a neighbor to the center of c_1 in the Delaunay graph $\mathbb{D}$ of the center point of circles. However, the Voronoi cell of c_3, computed using $C_3 = \{c_1, c_2, c_4, c_7\}$, in Figure 6(b) shows c_1 and c_2 are neighbors (as the Voronoi edge between c_1 and c_2 and the Voronoi edge between c_1 and c_3 intersects to form a vertex). This way, computation of a Voronoi cell of c_i (using N_i^*) can be used to update the neighborhood details of its neighbors.

5. Computation of initial Voronoi cell using a reduced touching disc method

Let $c_i \in C$, and C_i be the set of circles selected to compute an initial Voronoi cell, V_i^*. Let c_i for which Voronoi cell is computed be called the base circle. For the circles c_i in Figure 7, $C_i = \{c_1, c_2, c_3, c_4, c_5, c_6\}$. Initially, antipodal discs (ADs) between c_i and all the circles in C_i are computed and sorted as a list, $\mathbb{L}_i$, in the non-decreasing order of their radii. Now, each of the ADs in the list $\mathbb{L}_i$ are processed for emptiness in the order in which they appear in the list $\mathbb{L}_i$. Let AD_{i1} be of the smallest radius, and hence, it is empty. DE_{i1} and DE_{1i} are placed using the footpoints of AD_{i1}.

Lemma 1. *It is enough to check the intersection with at most two circles to check the emptiness of AD_{ij}.*

Proof. For checking the emptiness of AD_{ij}, it is enough to check the intersection with $c_k \in C_i$ such that AD_{ik} is less than AD_{ij}. While computing the Voronoi cell of a circle c_i, DEs are introduced in the order of radii of the associated disc of R_{min} (Figure 7(a)-(f)). This disc of R_{min} can be an empty AD or an empty TTD (DE_{i3} in Figure 7(d)). There can be at most two circles that can be identified by parametrically

Algorithm 1 *InitialVCells*($CircleSet$)

Compute the Delaunay graph $\mathbb{D}$ of the centers of the input circles.
For each circle $c_i \in C$, initialize C_i as the set of circles for which centers are neighbors to the center of c_i in $\mathbb{D}$.
Let S be the set of circles for which initial Voronoi cells are to be computed. Initialize $S = C$
while S is not empty **do**
  for each circle $c_i \in S$ **do**
    Let $\mathbb{NP}_i$ be the set of pairs of circles between which Voronoi edges exist. Initialize $\mathbb{NP}_i = \varnothing$.
    Initialize $\mathbb{L}_i$ as the list of antipodal disc between c_i and $c_j \in C_i$ sorted in non-decreasing order of radius.
    Considering the set of circles $C_i^* = C_i \cup c_i$, the first antipodal disc AD_{ij} in the list $\mathbb{L}_i$ is empty.
    Add DE_{ij} and DE_{ji} in the list of DEs, $\mathbb{DE}_i$.
    Remove the first AD from the list $\mathbb{L}_i$.
    while $\mathbb{L}_i$ is not empty **do**
      $ProcessAD(\mathbb{L}_i, \mathbb{DE}_i, c_i, C_i)$
      $ProcessTTD(\mathbb{L}_i, \mathbb{DE}_i, c_i, C_i, \mathbb{NP}_i)$
    end while
  end for
  Initialise $S^* = \varnothing$.
  for each $c_i \in S$ **do**
    for each entries of $\mathbb{NP}_i$ (Let (c_k, c_j) be one of the entries in $\mathbb{NP}_i$) **do**
      if $c_k \notin C_j$ **then**
        $C_j = C_j \cup c_k$ and $C_k = C_k \cup c_j$. $S^* = S^* \cup c_j \cup c_k$
      end if
    end for
  end for
  $S = S^*$
end while

Algorithm 2 $ProcessAD(\mathbb{L}_i, \mathbb{DE}_i, c_i, C_i)$

Pick the first disc in $\mathbb{L}_i$.
if the disc is a AD (say AD_{ij}) **then**
  Find C_D as the set of circles connected by DEs that are parametrically close to the footpoints of AD_{ij} on c_i.
  if (circles in C_D do not intersect AD_{ij}) **then**
    Add DE_{ij} and DE_{ji}.
    Check for parametrically closest DEs to form TTDs for both DE_{jk} and DE_{kj}.
    Insert TTDs, if any, to $\mathbb{L}_i$ maintaining non-decreasing order of radii.
  end if
  if ($c_k \in C_D$ intersects AD_{ij}) **then**
    Construct TTD_{ijk} and TTD_{kji}, if exist, and insert into $\mathbb{L}_i$ maintaining non-decreasing order of radii.
  end if
end if
Remove the disc from $\mathbb{L}_i$.

close DEs to the footpoint of an AD. If these two are not intersecting AD_{ij}, other circles in C_i will also not intersect AD_{ij}. □

If the consecutive DEs, DE_{ij} and DE_{ki} are found parametrically close, TTD between circles c_i, c_j and c_k are formed. Such TTDs are also inserted into the list $\mathbb{L}_i$, maintaining the order of radii. During the emptiness check of an AD, if a circle c_k is found intersecting AD_{ij}, TTD of the circles c_i, c_j and c_k are computed and inserted into the list $\mathbb{L}_i$, maintaining the order of radii. Hence, TTDs are formed under two situations: one is on identifying parametricaly close DEs and the second is on finding a circle intersecting an AD. When the turn of a TTD (TTD_{ijk}) arrives, it is considered as empty if either one or both of DE_{ij} and DE_{ki} are available. If only DE_{ij} is available, DE_{ik} is inserted using the footpoints of TTD_{ijk} on c_i and c_k. On finding a TTD as empty, Voronoi segments of its DEs are computed, and those DEs are deleted. It may

Algorithm 3 *ProcessTTD*$(\mathbb{L}_i, \mathbb{DE}_i, c_i, C_i, \mathbb{NP}_i)$

```
Pick the first disc from L_i.
if the disc is a TTD (say TTD_ijk) then
    if DE_ij and DE_ki exist then
        Compute Voronoi segments for DE_ij and DE_ki.
        Delete DE_ij and DE_ki
        NP_i = NP_i ∪ (c_j, c_k).
    else if only one DE (say DE_ij) exists then
        Compute Voronoi segments for DE_ij.
        Delete DE_ij.
        NP_i = NP_i ∪ (c_j, c_k).
        Add DE_ik.
        For DE_ik, identify parametrically closest DEs for computing TTDs.
        Insert TTDs, if any, appropriately in L_i.
    end if
    Remove the disc from L_i.
end if
```

be noted that in the modified touching disc method (hereafter, we call it a reduced touching disc method), ADs are computed only with the base circle and each neighboring circle. For the DEs, one end is directed either to or from the base circle.

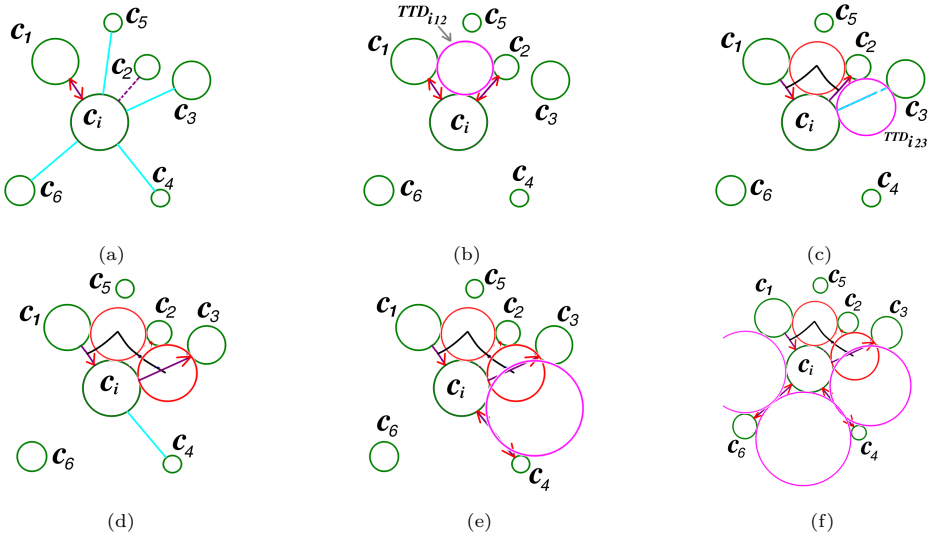
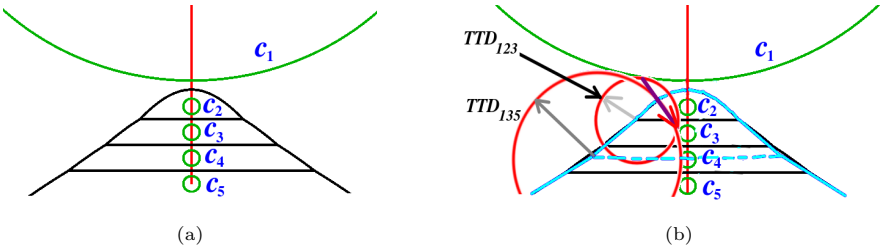


Figure 7: Reduced touching disc method for the computation of the Voronoi cell of a circle, c_i, using a set of circles as neighbors, is explained. (a) ADs between c_i and each of the remaining circles are computed and sorted in the order of radii. As AD_{i1} is of the smallest radius, it is empty. DE_{i1} and DE_{1i} are placed using the footpoints of AD_{i1}. (b) $ADi2$ is the next in the list of ADs. $ADi2$ is empty as c_2 is not intersecting it. DEs are placed and TTD_{i12} is computed and inserted into the list. (c) As TTD_{i12} is next in the list, it is processed and found empty as its DEs (DE_{i1} and DE_{2i}) exist; these DEs are deleted after computing its Voronoi segments. TTD_{i23} is computed as c_2 is intersecting AD_{i3}. (d) TTD_{i23} is processed and found as empty as DE_{i2} exists. DE_{i2} is deleted after computing its Voronoi segments. DE_{i3} is introduced using the footpoints of TTD_{i23}. (e) AD_{i4} is empty as c_3 and c_1 are not intersecting it. TTD_{i34} is introduced. (f) AD_{i5} is not empty as its footpoint on c_i is used for computing a Voronoi segment. AD_{i6} is empty as c_4 and c_1 are not intersecting it. TTD_{i46} and TTD_{i61} are introduced while processing AD_{i6}.

Lemma 2. *Let Algorithm 1 compute the Voronoi cell V_i^* for a circle c_i using a set of circles C_i, and N_i^* is the set of circles between which c_i shares Voronoi edges that form the boundary of V_i^*. If $V_i^{'}$ is the correct Voronoi cell corresponding to $VD(c_i \cup N_i^*)$ and doesn't have multi-edges between a pair of circles, Voronoi*



(a) (b)

Figure 8: Voronoi cells of circles computed using algorithm 1. (a) Delaunay graph (in red) of the centers of circles. Algorithm 1 correctly computes the Voronoi cells corresponding to circles c_2, c_3, c_4, and c_5. (b) Voronoi cell of c_1 is shown firm lines in cyan. Algorithm 1 doesn't create TTD_{134} as TTD_{123} is not processed and DE_{13} is not available while processing AD_{14}.

cell V_i^* is correct for $VD(c_i \cup N_i^*)$.

Proof. In Algorithm 1, $c_j \in N_i^*$ are identified by DEs between c_i and c_j which are introduced during the execution of the algorithm. The portion of c_i, used for computing a Voronoi segment, and between the footpoint of the *disc of* R_{max} and *disc of* R_{min}, is not used for computing another Voronoi segment. DEs always ensure the formation of TTDs such that all points along c_i will be used for computing Voronoi segments except in the case of Voronoi segments extending to infinity where the formation of TTD is not possible. This way, Voronoi cell V_i^* is a lower envelope of bisectors between c_i and $c_j \in N_i^*$, and V_i^* is correct for $VD(c_i \cup N_i^*)$.

□

For the second iteration onwards, Voronoi cells of a circle will be computed if its neighbors are updated after the previous iteration. For a given circle, for an extreme case, its list of neighbors may keep on updated until all neighbors are added to its list of neighbors. Since no further addition is possible, its Voronoi cell will not be computed again. This argument shows that Algorithm 1 will terminate.

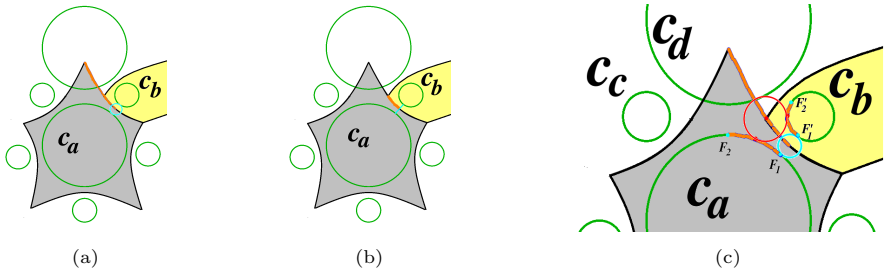
Figure 5(b) and (c) show Voronoi cells of c_1 and c_2, respectively, computed using the initial sets of circles obtained from the Delaunay graph of centers of the circles. Neighbors of c_1 and c_2 are updated while computing the Voronoi cells of c_3 and c_4. Using the updated list of neighbors, Algorithm 1, computes the Voronoi cell of c_1 and c_2 as shown in Figure 5(a).

In Figure 8(a), Algorithm 1 correctly computes Voronoi cells corresponding to c_2, c_3, c_4, and c_5. However, the Voronoi cell corresponding to c_1 is not computed correctly by algorithm 1 (Figure 8(b)). While computing the Voronoi cell of c_1, TTD_{134} is supposed to be created when processing AD_{14}. Since $TTD_{123} > AD_{14}$, DE_{13} is not available while processing AD_{14}. Hence, TTD_{134} will not be created by Algorithm 1. However, Figure 8(a) shows that the Voronoi cell of c_1 can be identified by the Voronoi cells of the remaining circles. The following section explains a topology matching procedure using TTDs that rectifies the incorrect Voronoi cells.

6. Topology matching

Consider Voronoi cells V_a^* and V_b^* corresponding to circles c_a and c_b, respectively. Let $c_a \in N_b^*$ and $c_b \in N_a^*$. In V_a^*, consider a Voronoi segment between c_a and c_b (Let this be VS_1)(Figure 9(a)). For V_b^*, a Voronoi segment between c_a and c_b exists and let this be VS_2 (Figure 9(b)).Consider the loci of footpoints traversed by the Voronoi disc of VS_1 along c_a. Let F_1 and F_2 be the footpoint of the *disc of* R_{min} and *disc of* R_{max}, respectively, corresponding to VS_1, on c_a (Figure 9(c)). Let F_1' and F_2' be the footpoint of the *disc of* R_{min} and *disc of* R_{max} of VS_1 on c_b. In the Figure we can see that *disc of* R_{min} for both VS_1 and VS_2 are the same. Now, we can visualize a Voronoi disc growing from the *disc of* R_{min} of VS_1, and at a certain instant it just touches c_d and the Voronoi disc coincides with *disc of* R_{max} of VS_2. If the Voronoi disc grows further, it starts intersecting c_d, which is not in N_a^*. As TTD_{acb} also intersects c_d, we have to include c_d in N_a^*. Hence, by adding TTD_{adb}, TTD list of c_a is updated.

Above argument is applicable to a Voronoi disc reducing to smaller radius from *disc of* R_{max} .



(a) (b) (c)

Figure 9: Basic idea of topology matching. (a) A Voronoi segment (VS_1) in V_a^* between c_a and c_b is highlighted in orange. (b) A Voronoi segment (VS_2) in V_b^* between c_a and c_b is highlighted in orange. (c) Footpoint of TTD_{adb} on c_a is nearer to F_1 than the footpoint of TTD_{acb} on c_a.

Algorithm 4 *TopologyMatching*(*CircleSet*)

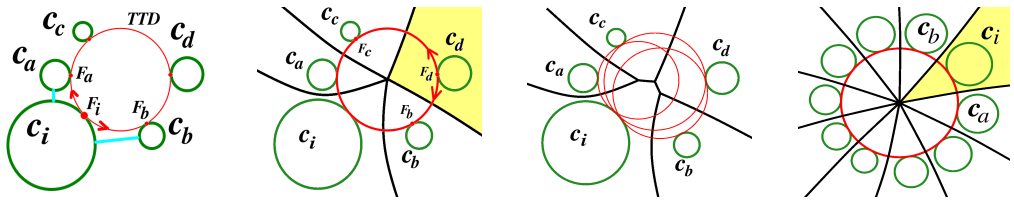
1: *InitialVCells*(*CircleSet*)
2: Initialize $S_t = C$
3: **while** S_t is not empty **do**
4: Initialize $S_t^* = \varnothing$
5: **for** each V_i^* of $c_i \in S_t$ **do**
6: Let $c_j \in \mathbb{NP}_i$ and AD_{ij} be of smallest radius, and its footpoint on c_i be F_i and that on c_j be F_i'.
7: Get the TTD next to F_i along c_i in CCW direction (Let this be TTD_{ilr}).
8: Get the TTD next to F_i' along c_j in CW direction (Let this be $TTD_{jl'r'}$).
9: Among $TTD_{ijl'}$, $TTD_{ijr'}$, TTD_{ijl} and TTD_{ijr}, the one with footpoint on c_i parametrically close to F_i in CCW is labelled as TTD_{ijn} and update TTD list of c_i.
10: Assign F_i as the footpoint of TTD_{ijn} on c_i and F_i' as the footpoint of TTD_{ijn} on c_n.
11: **if** F_i is crossing the footpoint of the smallest AD on c_i in CCW direction **then**
12: Exit the for loop
13: **end if**
14: **if** TTD_{ijn} is not in the TTD list of c_j **then**
15: Update TTD list of c_j
16: $S_t^* = S_t^* \cup c_j$
17: **end if**
18: **if** TTD_{ijn} is not in the TTD list of c_n **then**
19: Update TTD list of c_n
20: $S_t^* = S^* \cup c_n$
21: **end if**
22: Assign $j = n$ and go to step 7.
23: **end for**
24: $S_t = S_t^*$
25: **end while**

7. The algorithm and the parallel framework

Algorithm 1 computes the initial Voronoi cells, V_i^* of $c_i \in C$, following a reduced touching disc method that uses Algorithms 2 and 3 as subroutines. V_i^* holds the list of TTDs (that is considered empty corresponding to $C_i^* = N_i^* \cup c_i$, where N_i^* is the list of circles that are neighbors to c_i in V_i^*) sorted in the order of their footpoints on c_i. V_i^* is corrected by Algorithm 4, following a topology matching procedure. Voronoi cells of c_i for which Voronoi segments extend to infinity, Algorithm 4 repeats from the footpoints of minimum AD following a clockwise direction along c_i. For this, CCW and CW directions in Algorithm 4 have to be reversed.

The for loop that occurs on Line 5 in Algorithm 1 and the for loop in Algorithm 4 are implemented in parallel using a multi-core CPU framework with no data sharing between processors. For the ease of stating the algorithm, updating the TTD list of circles and S_t^* are shown inside the for loop of Algorithm 1. However, for parallel implementation, these are done outside the for loop to avoid the issues related to data racing.

7.1. Handling of input in non-general position



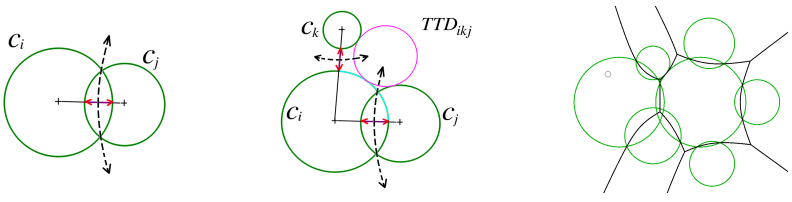
(a) (b) (c) (d)

Figure 10: Computing Voronoi cell for the non-general position input. (a) Parametric closeness along the TTD is considered for identifying circles contributing non-zero length Voronoi segment to the Voronoi cell of c_i. (b) F_c and F_b are parametrically close footpoints to F_d. Voronoi cell of c_d is highlighted in yellow. (c) Input in Figure (b) is slightly perturbed such that there is a non-zero length Voronoi segment between c_i and c_c and similarly between c_i and c_d. (d) All input circles are tangential to a common reference circle (in red). The Voronoi cell of c_i is highlighted in yellow. No circle other than c_a and c_b has a non-zero length Voronoi segment with c_i.

With a non-general position input, TTDs with the same foot point on c_i may be created for combinations of three circles. In the Figure 10(a), TTD_{iab}, TTD_{iac}, and TTD_{idb} are created making the same footpoint F_i on c_i. Circles with footpoints that are parametrically close to F_i along the TTD only contribute to non-zero length Voronoi segments (Figure 10(b)). When the input set in Figure 10(b) is slightly perturbed, as shown in Figure 10(c), TTDs with different footpoints on c_i are formed for TTD_{iab}, TTD_{iac}, and TTD_{idb} (red). In Figure 10(d), all combinations of three circles give the same TTD (red). While Algorithm 1 computing Voronoi cell of c_i, it uses footpoint details from TTDs to identify c_a and c_b that contribute non-zero length Voronoi segment with c_i. Voronoi cell with zero-length Voronoi segment with c_i are recorded in the TTD details corresponding to the footpoint F_i.

7.2. Handling of intersecting and hidden circles

For two intersecting circles, radius of AD is computed as a negative value and DEs are defined using the footpoints (Figure 11(a)). Parametric close and consecutive DEs are used to compute TTDs (Figure 11(b)), and possibility of an internal TTD is also checked for computing a Voronoi vertex (Figure 11(c)). A hidden circle (a circle that is completely contained in another circle) can be identified while computing ADs using the value of its radius.



(a) (b) (c)

Figure 11: Voronoi diagram of intersecting circles (a) Footpoints of DEs (DE_{ij} and DE_{ji}) are computed using the line passing through the centers of the circles. (b) DE_{ji} and DE_{ij} form TTD_{ikj}. The portion of circle, c_i used for computing Voronoi segments while processing TTD_{ikj} is highlighted in cyan color. (c) A hidden circle is highlighted in grey color. For intersecting circles, the possibility of an internal TTD is also checked to complete the Voronoi diagram.

7.3. Complexity of the algorithm

The complexity of the algorithm used for computing the Delaunay graph is $O(n \log n)$. As the Delaunay graph is planar, the total number of ADs computed for the initial Voronoi cells is linear in n. The same is true for TTDs, as parametric closeness is used to compute it. In a worst-case where a circle has the remaining circles in the input set as it neighbors, the complexity of finding the initial Voronoi cells is $O(n log n)$. Hence, in the first stage of iteration, complexity in an iteration in which Voronoi cell of all circles are computed is $O(n \log n)$. Voronoi cell corrections in the second stage involve searching three touch discs (TTDs) from a list, sorted in the order of footpoints. These TTDs correspond to vertices in the Voronoi cells. In extreme cases, the number of vertices of a few Voronoi cells can be $O(n)$. However, the total number of vertices in a Voronoi diagram is also $O(n)$. Hence, for an extreme case, the complexity of the second stage is $O(n \log n)$. However, for uniformly distributed neighbors, complexity of first and second stage iterations becomes $O(n)$.

7.4. Correctness of the algorithm

The Voronoi cell, V_i^* constructed at the end of Algorithm 1 may not be correct considering the actual Voronoi diagram, $VD(C)$. However, it is correct for a small subset of input circles (Lemma 2). The exemption for multi-edge Voronoi cells will be addressed by Algorithm 4, where these multi-edge Voronoi cells are also corrected by those Voronoi cells of its neighbors that are free from multi-edges. Topology matching is done using TTDs and its footpoints. As the centers of these TTDs correspond to Voronoi vertices, and matching occurs from vertex to vertex, edges also get matched over iterations.

Lemma 3. *The Voronoi diagram computed by Algorithm 4 is correct so that the computed Voronoi cells are disjoint, and adjacent/nonadjacent Voronoi cells in $VD(C)$ remain adjacent/nonadjacent in the computed Voronoi diagram.*

Proof. Algorithm 4 processes a Voronoi cell V_i^* for which N_i^* is the set of neighbors and traverses the boundary of the Voronoi cell from vertex to vertex by constructing the Voronoi edges connecting it. New matching circles are verified by constructing TTDs and checking emptiness of TTD by parametric closeness. New list of TTDs are used to define the latest N_i^*. Voronoi edge between c_i and $c_j \in N_i^*$ is computed by constructing bisectors between c_i and c_j. This Voronoi edges form the lower envelope of bisectors between c_i and $c_j \in N_i^*$ so that the latest V_i^* is also correct for $VD(c_i \cup N_i^*)$. Voronoi cell V_j^* of $c_j \in N_i^*$ and V_i^* are disjoint at the end of Algorithm 4. Hence, at the end of Algorithm 4, all Voronoi cells are disjoint.

Let V_i be the Voronoi cell of c_i corresponding to $VD(C)$, and N_i be the set of neighboring circles such that V_i and V_j are adjacent Voronoi cells, where $c_j \in N_i$. If $c_m \in N_i \setminus N_i^*$, $V_i \subset V_i^*$ (Figure 5). However, at the end of Algorithm 4, if $c_m \notin N_i^*$, $c_i \notin N_m^*$; and this implies, $V_m \subset V_m^*$. However, this leads to $V_i^* \cap V_m^* \neq \emptyset$. As it contradicts the set of disjoint regions of Voronoi cells, $N_i \setminus N_i^* = \emptyset$; implies, adjacent Voronoi cells in $VD(C)$ remain adjacent in the computed Voronoi diagram.

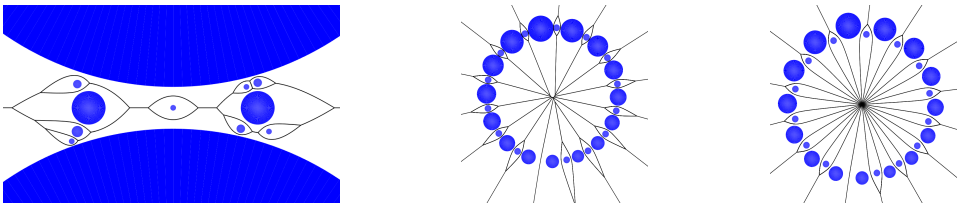
Now, if $c_m \in N_i^* \setminus N_i$, Voronoi edge between c_m and c_i forms a boundary for V_i^*, and leads to $V_i \subset V_i^*$. However, if $c_m \in N_i^*$, $c_i \in N_m^*$, and hence, $V_m \subset V_m^*$, implies, $V_m^* \cap V_i^* \neq \emptyset$. As it contradicts the set of disjoint Voronoi cells, $N_i^* \setminus N_i = \emptyset$; implies nonadjacent Voronoi cells in $VD(C)$ remain nonadjacent in the computed Voronoi diagram. Thus, $N_i = N_i^*$, and the lemma. □

8. Results and discussions

We implemented Algorithm 4 in C++ using multi-threading and run on a PC with: Intel Core i5-2400 CPU @ 3.10GHz × 4; 8GB RAM; Ubuntu 18.04.5 LTS [64bit]. We used The CGAL Project (2020) to compute the Delaunay triangulation of the center points of circles and the algorithm of Gavrilova and Rokne (2003) for constructing TTDs. Voronoi edges are exactly computed using the bisector equation in Hu et al. (2017).

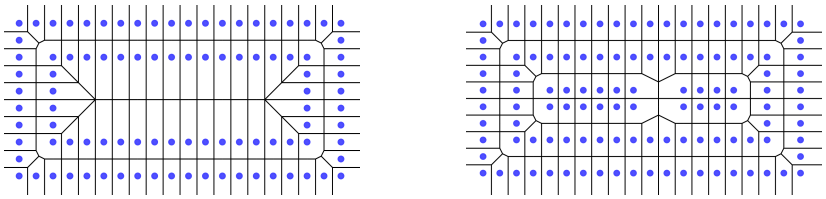
Figure 12(a) shows the Voronoi diagram for a test case given by Lee et al. (2016), where multi edges are present in Voronoi cells of four circles. The topology of these cells is corrected by neighboring Voronoi cells that do not have multiple edges. Figure 12(b) is the Voronoi diagram for a set of circles for which centers are on the circumference of a common reference circle. Here, as the circles are not of uniform radius, there

is not a common vertex for all the circles. In Figure 12(c), all circles in the input set are tangential to a common reference circle. Here, all circles in the input set are supposed to have a common reference circle, but the deviation is due to the error in generating the input set and constructing TTDs. Figure 13 is also an examples of input in non-general position, where centres of uniform radii circles are placed at grid points. Figure 14(a) shows Voronoi diagram of a set of 1000 random generated circles, and Figure 14(b) shows a zoom-in of the boxed region.



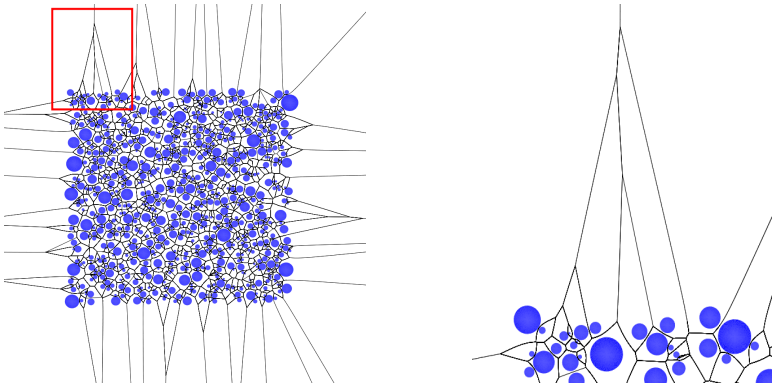
(a) (b) (c)

Figure 12: Voronoi diagram for: (a) a configuration given in Lee et al. (2016); (b) a set of circles for which centres are on the circumference of a common reference circle; and (c) a set of circles which are tangential to a common reference circle.



(a) (b)

Figure 13: Voronoi diagram of identical circles placed at grid points (test cases given in Lee et al. (2016) for input in non-general position).



(a) (b)

Figure 14: (a) Voronoi diagram of a set of 1000 circles and (b) a zoom-in of the boxed region.

The touching disc method of Sundar et al. (2020) requires ADs between all pairs of circles and thereby leads to an $O(n^2)$ complexity. The dynamic sampling method of Mukundan and Muthuganapathy (2021) reduces the computational complexity to $O(n\ log\ n)$ by employing the Delaunay graph of sample points selected following a dynamic sampling approach. However, this approach is sequential and cannot be made parallel. In the proposed algorithm, a Voronoi cell-wise computation is followed, and for each cell, ADs are not computed between the neighboring circles. This largely reduced the computational time and complexity

and extended the possibility of parallel computing. Moreover, for computing each Voronoi cell, a small subset of input circles is used, and hence, the memory requirement for computing each Voronoi cell is less in parallel computing. This helps to implement the algorithm using separate memory for each processor. The proposed method is robust as it can handle input in non-general position as in the case of Lee et al. (2016) and Mukundan and Muthuganapathy (2021). GPU based algorithms of Hu et al. (2017); Li et al. (2019) compute bisector between all pairs of circles, and hence the algorithmic complexity is of $O(n^2)$. For our algorithm, the total memory required for a randomly generated input set varies linearly with input size (Figure 15(a)).

In Figure 15(a): Dyn-samp-appr represents dynamic sampling approach of Mukundan and Muthuganapathy (2021); CGAL-filtered-exact-hierarchy represents the algorithm of Devillers (1998) that maintains a hierarchy of Voronoi diagram to accelerate an incremental construction of Voronoi diagram; Topol-match-1 represents topology matching algorithm implemented with a single thread; and Topol-match-4 represents topology matching algorithm implemented with four threads. In CGAL library, CGAL-exact is the implementation of Karavelas and Yvinec (2002) with exact number type which takes more than 300 seconds for a circle set of size 5×10^4. Even though we are using a topology matching, an exact computation is followed for computing the Voronoi diagram. It can be seen that for a set of 5×10^5 circles, Topol-match-4 takes less than 14 seconds. Our algorithm works well for a set of 1×10^6 circles with Topol-match-1 and Topol-match-4 taking 28.2 seconds and 66.3 seconds, respectively.

Figure 15(b) shows the effect of multithreading for our algorithm corresponding to compute the Voronoi diagram for a randomly generated circle set of size 5×10^5. The computation time is reduced to less than half of the single thread implementation with four threads. The overhead of the sequential part of the algorithm consists of computing the Delaunay graph of the center point of circles, assigning the initial set of neighbors to each circle, updating each subset that is used for computing a Voronoi cell, and updating the TTD details. This is obtained as approximately 14% of the total time taken by Topol-match-1 for all sets of randomly generated circles. As updating occurs outside the threads, there is no communication required between the threads. Synchronization of threads depends on the size of the subset of circles used for computing each Voronoi cell. There can be input configurations where a circle $c_i \in C$ can have all the remaining circles in C included in the initial subset of circles used for computing the Voronoi cell of c_i. Such configurations affect the synchronization and the performance of the algorithm. However, the algorithm gives correct results for such extreme configurations also.

Our algorithm divides the problem into a number of sub-problems equal to the size of the input set. Even though this makes the scaling easier, a detailed study on scalability requires extending this to a GPU-based algorithm, which we kept as future work. As the size of each subset is decided by the Delaunay graph of center points of circles, it gives the correct result for uneven distribution of input sets, and the total memory requirement is $O(n)$. The parallel algorithms of Hu et al. (2017); Li et al. (2019) require samples and associated details corresponding to each Voronoi cell and requires $O(n^2)$ memory as the size of each of their sub-problem is of $O(n)$. More memory requirement combined with $O(n^2)$ complexity makes their algorithm more computationally expensive for a large input set (The results showing the performance of their algorithm does not consider input size beyond 5000. Our algorithm computes the Voronoi diagram of 1 million circles with a total memory requirement of 4.3 GB). Figure 15(c) shows the convergence of first stage and second stage iterations. In both stages, the total number of circles processed are less than $2n$. The number of circles processed after the first iteration is negligible for the second stage. That shows that Voronoi cells of most of the circles are computed correctly in the first stage of iteration using the reduced touching disc method. Convergence of first stage iteration shows, half of the input circles are processed in the second iteration, and that shows neighborhood details of half of the circles are changed when shifted from points to circles.

8.1. Limitations and future works

The computational time is more for an input set where any circle has neighbors of size in the $O(n)$. However, the algorithmic complexity is still $O(n \log n)$, and even sequential computing gives a decent time performance. However, the scope of this method lies in exploring the possibility of computing the Voronoi diagram for closed planar curves with $O(n \log n)$ complexity. Hence, the future work is to develop an

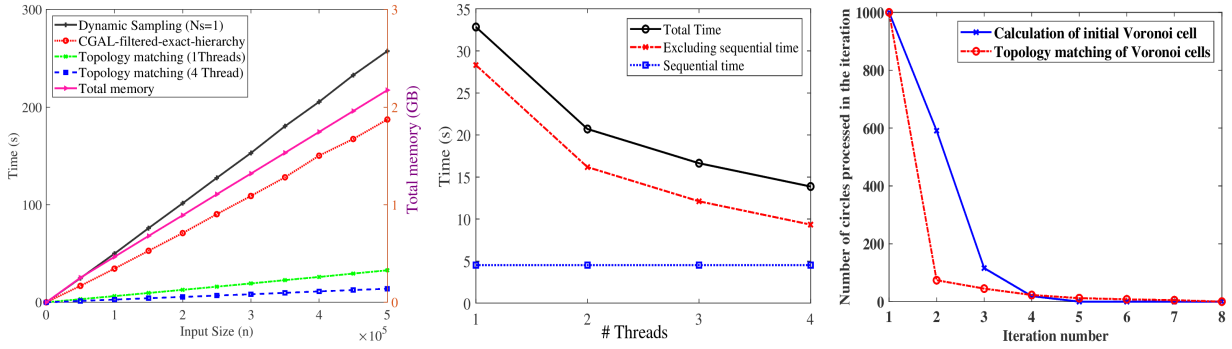


Figure 15: Results for random circles. (a) Computation time comparison with CGAL-filtered-exact-hierarchy and dynamic sampling approach. (b) Variation of computational time with number of threads for a set of 5×10^5 circles. (c) Convergence of first and second stage iteration for a set of 1000 circles. Number of of circles processed in each iteration.

$O(n \log n)$ algorithm for computing the Voronoi diagram of a set of planar curves using the reduced touching disc method combined with topology matching. Moreover, implementation of the parallel framework is currently CPU-based. GPU-based implementation is also in the pipeline. Extending this method to compute the Voronoi diagram of a set of spheres is also considered as a future work.

9. Conclusions

We have presented a novel algorithm for computing the Voronoi cells of circles using parallel framework. The independent nature of the computation of the Voronoi cells using only a small subset of input circles has helped us to exploit the multi-core computation of the CPU. The average number of circles used for computing a Voronoi cell is considerably less. This has helped to avoid data sharing and reduce the amount of memory required for each processor, an essential consideration for achieving greater computational performance and handling a large input set. The touching disc has been instrumental in finding branch points without using any bisector intersection. A reduced touching disc method combined with a topology matching procedure has been the key point in reducing the complexity from $O(n^2)$ to $O(n \log n)$ and this helped in efficiently handling a large input set. Even with a four-core CPU, the algorithm computes the Voronoi diagram of 5×10^5 circles in less than 14 seconds. In addition, efficacy in handling input in non-general position and intersecting circles has added robustness to the proposed algorithm. Possible future works have also been mentioned.

References

Aggarwal, A., Chazelle, B., Guibas, L., Ó'Dúnlaing, C., Yap, C., 1988. Parallel computational geometry. Algorithmica 3, 293–327.

Anton, F., Mioc, D., Gold, C., 2009. The Voronoi diagram of circles and its application to the visualization of the growth of particles, in: Transactions on Computational Science III. Springer, pp. 20–54.

Aurenhammer, F., 1991. Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Computing Surveys (CSUR) 23, 345–405.

Bernaschi, M., Lulli, M., Sbragaglia, M., 2017. GPU based detection of topological changes in Voronoi diagrams. Computer Physics Communications 213, 19–28.

Cao, T.T., Nanjappa, A., Gao, M., Tan, T.S., 2014. A GPU accelerated algorithm for 3D Delaunay triangulation, in: Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 47–54.

Cornea, N.D., Silver, D., Min, P., 2007. Curve-skeleton properties, applications, and algorithms. IEEE Transactions on Visualization and Computer Graphics 13, 530.

Devillers, O., 1998. Improved incremental randomized Delaunay triangulation, in: Proceedings of the fourteenth annual symposium on Computational geometry, pp. 106–115.

Devillers, O., Karavelas, M., Teillaud, M., 2015. Qualitative symbolic perturbation: a new geometry-based perturbation framework. INRIA , 34.

Edelsbrunner, H., Mücke, E.P., 1990. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. ACM Transactions on Graphics (tog) 9, 66–104.
Emiris, I.Z., Karavelas, M.I., 2006. The predicates of the Apollonius diagram: algorithmic analysis and implementation. Computational Geometry 33, 18–57.
Gavrilova, M.L., Rokne, J., 2003. Updating the topology of the dynamic Voronoi diagram for spheres in Euclidean d-dimensional space. Computer Aided Geometric Design 20, 231–242.
González, R.E., 2016. PARAVT: Parallel Voronoi tessellation code. Astronomy and Computing 17, 80–85.
Goodrich, M.T., Ó'Dúnlaing, C., Yap, C.K., 1993. Constructing the Voronoi diagram of a set of line segments in parallel. Algorithmica 9, 128–141.
Hu, Z., Li, X., Krishnamurthy, A., Hanniel, I., McMains, S., 2017. Voronoi cells of non-general position spheres using the GPU. Computer-Aided Design and Applications 14, 572–581.
Jin, L., Kim, D., Mu, L., Kim, D.S., Hu, S.M., 2006. A sweepline algorithm for Euclidean Voronoi diagram of circles. Computer-Aided Design 38, 260–272.
Karavelas, M.I., Yvinec, M., 2002. Dynamic additively weighted Voronoi diagrams in 2D, in: European Symposium on Algorithms, Springer. pp. 586–598.
Kim, D.S., Kim, D., Sugihara, K., 2001a. Voronoi diagram of a circle set from Voronoi diagram of a point set: I. Topology. Computer Aided Geometric Design 18, 541–562.
Kim, D.S., Kim, D., Sugihara, K., 2001b. Voronoi diagram of a circle set from Voronoi diagram of a point set: II. Geometry. Computer Aided Geometric Design 18, 563–585.
Lee, D.T., Drysdale, III, R.L., 1981. Generalization of Voronoi diagrams in the plane. SIAM Journal on Computing 10, 73–87.
Lee, M., Sugihara, K., Kim, D.S., 2016. Topology-oriented incremental algorithm for the robust construction of the Voronoi diagrams of disks. ACM Transactions on Mathematical Software (TOMS) 43, 1–23.
Li, X., Krishnamurthy, A., Hanniel, I., McMains, S., 2019. Edge topology construction of Voronoi diagrams of spheres in non-general position. Computers & Graphics 82, 332–342.
Liu, X., Ma, L., Guo, J., Yan, D.M., 2020. Parallel computation of 3D clipped Voronoi diagrams. IEEE Transactions on Visualization and Computer Graphics .
Mahboubi, H., Aghdam, A.G., 2013. An energy-efficient strategy to improve coverage in a network of wireless mobile sensors with nonidentical sensing ranges, in: 2013 IEEE 77th Vehicular Technology Conference (VTC Spring), pp. 1–5. doi:10.1109/VTCSpring.2013.6691875.
Mukundan, M.K., Muthuganapathy, R., 2021. A dynamic sampling approach towards computing Voronoi diagram of a set of circles. Computer Aided Geometric Design 90, 102023. URL: https://www.sciencedirect.com/science/article/pii/S0167839621000686, doi:https://doi.org/10.1016/j.cagd.2021.102023.
Peterka, T., Morozov, D., Phillips, C., 2014. High-performance computation of distributed-memory parallel 3D Voronoi and Delaunay tessellation, in: SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE. pp. 997–1007.
Ray, N., Sokolov, D., Lefebvre, S., Lévy, B., 2018. Meshless Voronoi on the GPU. ACM Transactions on Graphics (TOG) 37, 1–12.
Rong, G., Liu, Y., Wang, W., Yin, X., Gu, D., Guo, X., 2010. GPU-assisted computation of centroidal Voronoi tessellation. IEEE transactions on visualization and computer graphics 17, 345–356.
Rong, G., Tan, T.S., 2006. Jump flooding in GPU with applications to Voronoi diagram and distance transform, in: Proceedings of the 2006 symposium on Interactive 3D graphics and games, pp. 109–116.
Ryu, J., Lee, M., Kim, D., Kallrath, J., Sugihara, K., Kim, D.S., 2020. Voropack-d: Real-time disk packing algorithm using Voronoi diagram. Applied Mathematics and Computation 375, 125076.
Sugihara, K., 1992. A simple method for avoiding numerical errors and degeneracy in Voronoi diagram construction. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 75, 468–477.
Sugihara, K., 1993. Approximation of generalized Voronoi diagrams by ordinary Voronoi diagrams. CVGIP: Graphical Models and Image Processing 55, 522–531.
Sugihara, K., Iri, M., 1992. Construction of the Voronoi diagram for "one million" generators in single-precision arithmetic. Proceedings of the IEEE 80, 1471–1484.
Sugihara, K., Sawai, M., Sano, H., Kim, D.S., Kim, D., 2004. Disk packing for the estimation of the size of a wire bundle. Japan Journal of Industrial and Applied Mathematics 21, 259–278.
Sundar, B.R., Mukundan, M.K., Muthuganapathy, R., 2020. A unified approach towards computing Voronoi diagram, medial axis, Delaunay graph and α-hull of planar closed curves using touching discs. Computers & Graphics 89, 131 – 143. URL: http://www.sciencedirect.com/science/article/pii/S0097849320300613, doi:https://doi.org/10.1016/j.cag.2020.05.010.
The CGAL Project, 2020. CGAL User and Reference Manual. 5.2 ed., CGAL Editorial Board.
Wang, C., Roth, H.R., Kitasaka, T., Oda, M., Hayashi, Y., Yoshino, Y., Yamamoto, T., Sassa, N., Goto, M., Mori, K., 2019. Precise estimation of renal vascular dominant regions using spatially aware fully convolutional networks, tensor-cut and Voronoi diagrams. Computerized Medical Imaging and Graphics 77, 101642. URL: https://www.sciencedirect.com/science/article/pii/S0895611119300618, doi:https://doi.org/10.1016/j.compmedimag.2019.101642.
Yap, C.K., 1990. Symbolic treatment of geometric degeneracies. Journal of Symbolic Computation 10, 349–370.
Yuan, Z., Rong, G., Guo, X., Wang, W., 2011. Generalized Voronoi diagram computation on GPU, in: 2011 Eighth International Symposium on Voronoi Diagrams in Science and Engineering, IEEE. pp. 75–82.
Zheng, A., Bian, S., Chaudhry, E., Chang, J., Haron, H., You, L., Zhang, J., 2021. Voronoi diagram and Monte-Carlo simulation based finite element optimization for cost-effective 3D printing. Journal of Computational Science 50, 101301.

URL: https://www.sciencedirect.com/science/article/pii/S187775032100003X, doi:https://doi.org/10.1016/j.jocs.2021.101301.

Zivanic, M., Daescu, O., Kurdia, A., Goodman, S., 2012. The Voronoi diagram for graphs and its application in the Sickle Cell Disease research. Journal of Computational Science 3, 335–343. URL: https://www.sciencedirect.com/science/article/pii/S1877750311001001, doi:https://doi.org/10.1016/j.jocs.2011.10.006. Advanced Computing Solutions for Health Care and Medicine.