Conjectures concerning the geometry of 2-point Centroidal Voronoi Tessellations

Emma Twersky

May 2017

Abstract

This paper is an exploration into centroidal Voronoi tessellations, or CVTs. A centroidal Voronoi tessellation is defined, and we specifically focus on 2-point CVTs. We also explore creating visualizations of the algorithms which generate CVTs, including the implementations of MacQueen's and Lloyd's methods. These visualizations are written using p5.js, hosted online at `carrot.whitman.edu/CVT`. We explore the algorithms and prove a method to test point inclusion in a convex polygon using the intersection of half planes. Finally, we explore characteristics and properties of 2-point CVTs in order to develop conjectures about their structure, including examining the stability of these methods and variations of these conjectures using alternative distance metrics.

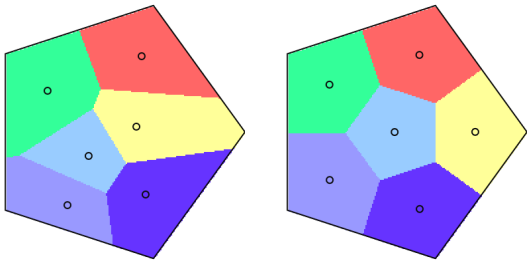
1 Introduction

We will introduce and define the characteristics of a centroidal Voronoi Tesselation, the primary focus of this exploratory paper. We aim to define these figures in order to then create technological applications which generate visualizations and help us conjecture upon their possible properties. We will specifically focus on the 2-point CVTs of regular polygons and under two common algorithms for their generation, then expand this to alternative distance metrics.

1.1 Centroidal Voronoi Tessellations

In order to look at centroidal Voronoi tessellations, or CVTs, we examine Voronoi tessellations and define the restrictions under which they become centroidal. Figure 1 shows an example of a 2-point VT and CVT on a regular 5-gon. The following definitions come from Du, Faber, and Gunzburger in [5].

Figure 1: Example Tesselations



(a) Voronoi Tesselation (b) Centroidal Voronoi Tesselation

Definition 1.1 (Voronoi Tessellations)**.** *Voronoi tessellations have an intuitive formation, given the following description. Take a connected, closed set in* $\mathbb{R}^2$*, called* Ω*, and take a finite set of points (called generators) in* Ω*. Around each generator* z_i *draw a "bubble"* V_i *(called the Voronoi region). Inside each* V_i *are all the points closer to* z_i *than, or as close to* z_i *as, any of the other points in the set. Points may be equidistant to two generators, therefore lying on the line bordering two Voronoi regions. The union of all of these regions is the original set.*

Definition 1.2 (Centroidal Voronoi Tessellations)**.** *A centroidal Voronoi tessellation (CVT) is a specific formation of a Voronoi tessellation. The additional restriction is that the generator points are also the centroids of their associated Voronoi region. That is, the balancing point of the region is at the generator. We will continue to look at CVTs, with specific focus on CVTs of two generator points (or 2-point CVTs) since they are much more regular looking, and have many additional properties and features that Vornoi tessellations do not have.*

1.2 p5$_*$js

p5.js is a JavaScript library which functions as a free implementation of Processing using JavaScript packages and libraries. In this paper, p5.js is the primary language used to create visualizations and sketches which generate and explore CVT properties and algorithms. These libraries can be referenced at `p5js.org`.

p5.js has online tutorials and directories, found at `p5js.org/tutorials/`.

My p5.js sketches will be hosted on the website `carrot.whitman.edu/CVT`, also using DropBox as a storage location for these files.

2 Generating CVTs with p5$_*$js

We will now use the definitions of CVTs to explore several algorithms which generate CVTs and implement these algorithms into p5.js applications.

2.1 Point Location Testing

In anticipation of creating a visualization for CVT generation, one issue that will arise is determining if a randomly generated point is within the given n-gon domain. To do this, we propose a location test for points that are intersections of half planes. It is much quicker for a computer to generate a random point within the circle inside of which the shape is inscribed. We then test if the given point is within the n-gon's domain by using an algorithm based on the following theorem.

Theorem 2.1. *Let A be a regular n-gon in the plane, with the vertices $P_0, P_2, \ldots P_{n-1}, P_n$, where $P_0 = P_n$, labeled in a counter-clockwise direction. Let Z be a point on the plane.*

Let D and E be the displacement vectors from the vertex P_i defined as $D_i = P_{i+1} - P_i = \overrightarrow{P_i P}_{i+1}$ and $E_i = Z - P_i = \overrightarrow{P_i Z}$. If $D_i \times E_i \cdot \hat{k} > 0$ for all $0 \leq i \leq n$, then Z is contained within A.

Proof. Consider each edge of the n-gon as a line on the plane, defined by the displacement vector D_i, passing through the point P_i. Since the vertices increase in a counterclockwise direction, we will define the set of points to the left of the orientation direction (line) as being inside the half plane. By the right hand rule, if a point is on the left side of the line, then $D_i \times E_i \cdot \hat{k} > 0$.

As concluded by Blackwell in [1], convex polygons are intersections of half planes. If a point is in a polygon, it follows that it is in the intersection of all half planes. Therefore the point is to the left of all half planes.

Therefore if $D_i \times E_i \cdot \hat{k} > 0$ for all $0 < i < n$, then Z is contained within A. □

A visual example of this test can be seen at `blackpawn.com/texts/pointinpoly/`. This version uses the full cross product, but we have simplified the code to focus solely on the sign of the cross product.

2.2 MacQueen's Algorithm

We now use Theorem 2.1 to begin building a visualization of CVTs in p5.js.

We use the vector package included in p5.js to determine if a randomized point within a circle is contained within the inscribed n-gon. This sketch can be seen at `carrot.whitman.edu/CVT/SupportingFeatures/DotLocationTest/`. We describe Mac-Queen's Algorithm for generating CVTs, as found in Du, Faber, and Gunzburger, [5], and use this algorithm so create a p5.js sketch which visualizes the algorithm's process for generating CVTs.

Definition 2.2 (MacQueen's Algorithm)**.** *Given a set Ω, a positive integer k, and a probability density function ρ defined on Ω,*

0. select an initial set of k points $\{z_i\}_{i=1}^k$, e.g., by using a Monte Carlo method or by user choice; set $j_i = 1$ for $i = 1, \ldots, k$;

1. *select a $y \in \Omega$ at random, according to the probability density function $\rho(y)$;*

2. *find the z_i that is closest to y; denote the index of that z_i by i^*;*

3. *set*
$$z_{i^*} \leftarrow \frac{j_{i^*} z_{i^*} + y}{j_{i^*} + 1} \qquad \text{and} \qquad j_{i^*} \leftarrow j_{i^*} + 1;$$
this new z_{i^*}, along with the unchanged $z_i, i \neq i^*$, forms the new set of points $\{z_i\}_{i=1}^k$

4. *if this new set of points meets some convergence criterion, terminate; otherwise, go back to step 1.*

MacQueen's Algorithm in p5.js

We then use Mac-Queen's algorithm and the dot location test sketch to generate CVTs in p5.js. This sketch can be found at `carrot.whitman.edu/CVT/MacQueen`.

2.3 Lloyd's Algorithm

An algorithm we hope will be more precise than MacQueen's, is Lloyd's. This algorithm takes more work and is therefore less efficient, but we explore it's definition and visual representations in the hopes of creating a more accurate p5.js CVT generator.

We describe Lloyd's Algorithm for generating CVTs, as found in Du, Faber, and Gunzburger, [5], and use this algorithm to create a p5.js sketch which visualizes the algorithm's process for generating CVTs.

Definition 2.3 (Lloyd's Algorithm)**.** *Given a set Ω, a positive integer k, and a probability density function ρ defined on Ω,*

0. *select an initial set of k points $\{z_i\}_{i=1}^k$, e.g., by using a Monte Carlo method or by user choice;*

1. *construct the Voronoi tessellation $\{V_i\}_{i=1}^k$ of Ω associated with the points $\{z_i\}_{i=1}^k$;*

2. *compute the mass centroids of the Voronoi regions $\{V_i\}_{i=1}^k$ found in step 1; these centroids are the new set of points $\{z_i\}_{i=1}^k$.*

3. *if this new set of points meets some convergence criterion, terminate; otherwise, go back to step 1.*

Lloyd's Algorithm in p5.js

We then use Lloyd's algorithm to generate CVTs in p5.js. This sketch can be found at `carrot.whitman.edu/CVT/Lloyds`. We now use this sketch to examine properties of CVTs and possible improvements in generating accurate CVTs.

2.4 Convergence of MacQueen's and Lloyd's Algorithms

We have already shown the psuedocode for MacQueen's and Lloyd's algorithms that we have implemented into p5.js applications. Now, before we state any further conjectures on the figures these applications generate, we must be sure that these algorithms, and therefore the applications, converge to CVTs.

We refer to Du for some discussion on the recent development of a general convergence theory for Lloyd's algorithm [3], this paper specifically works to conclude local convergence of Lloyd's algorithm within $\mathbb{R}^2$ on a uniform density plane. The only theoretical convergence proved is given for the one dimensional case. Since a more rigorous theoretical proof of convergence has not been found, we rely on the author's empirical investigation into run time of the algorithms, measuring movement of the generator points, in order to conclude that Lloyd's algorithm will converge. Because of this, the application shows that in simple examples, we can see that the algorithm visually converges to known CVT representations.

A similar approach is used to analyze the convergence of MacQueen's algorithm within finite local space. [3] These arguments are then summarized in an analysis of the convergence run time of each algorithm in [2]. This paper summarizes the findings that through numerical computation and optimization, both algorithms converge to a figure, representative of a CVT, in at best linear time, which increases as the number of generators increase. Thus, while not efficient for large problems, these references show the proposed convergence of these algorithms, thus we can continue to examine the application's outputs as CVTs and the possible properties they show.

3 Exploring Conjectures of 2-point CVTs

We will now use the application to conjecture upon possible properties and characteristics of CVTs, specifically focusing on the 2-point CVT of regular polygons. In the following section, we aim to provide examples and explain the following properties of 2-point CVTs. These examples are intended to be accompanied by the applications found at `carrot.whitman.edu/CVT/Lloyds`.

3.1 Stable CVTs for Lloyd's Method

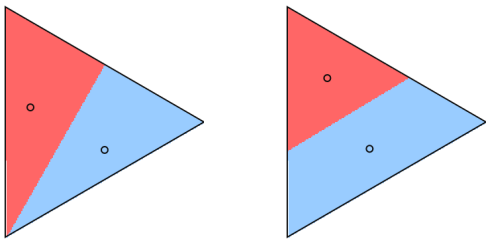
The following sections will specifically focus on defining and showing the stability of the CVTs which the Lloyd's application converges too.

We begin by examining n-gon of increasing sides to examine patterns of convergence of the figures.

3.1.1 Regular 3-gon

In any regular n-gon, the 2-point CVTs are not unique. In the following figure, we present a regular 3-gon and the known CVTs of the shape, as presented in [4].

Figure 2: Known CVTs of a regular 3-gon



(a) Known long boundary CVT of a regular 3-gon

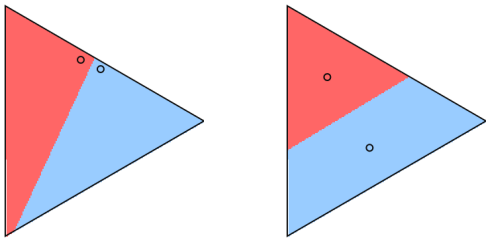
(b) Known short boundary CVT of a regular 3-gon

From this figure, we observe two possible CVTs the application may produce, up to rotational symmetry.

We now see if we can predict the CVT to which Lloyd's algorithm will converge.

Using the Lloyd's Algorithm application, we will construct an equilateral triangle, and test what happens after the program is run for a sufficient period of time so that the generator points no longer visibly move. First, we will place the two points side by side on one of the edges.

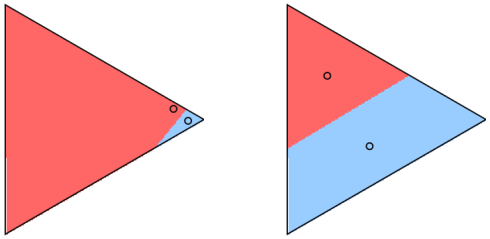
Figure 3: Exploration of Lloyd's on a regular 3-gon



(a) Before application is run

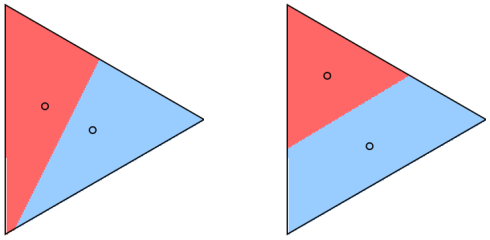
(b) After application is run

Next, we test when the points are placed side by side on of the vertexes.



(c) Before application is run (d) After application is run

We now test when the points are places as close as possible to the the long boundary CVT configuration.



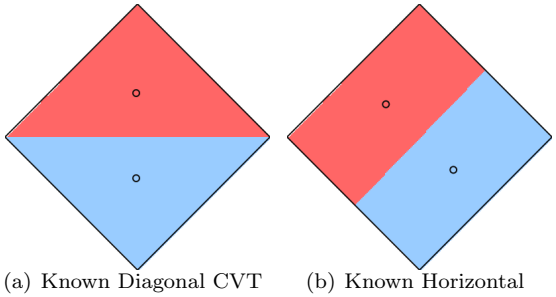
(e) Before application is run (f) After application is run

In Figure 3, we see that no initial placement of the two points results in the long boundary CVT configuration.

3.1.2 Regular 4-gon

We now construct parallel arguments using a square.

Figure 4: Known CVTs of a regular 4-gon



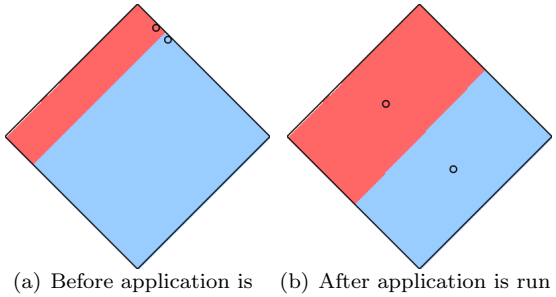
(a) Known Diagonal CVT of a regular 4-gon (b) Known Horizontal CVT of a regular 4-gon

From this figure, we observe two possible CVTs the application may produce, up to rotational symmetry.

We now see if we can predict the CVT to which Lloyd's algorithm will converge.

Using the Lloyd's Algorithm application, we will construct a four sided polygon, or a regular 4-gon, and test what happens after the program is run for a sufficient period of time so that the generator points no longer visibly move. First, we will place the two points side by side on one of the edges.

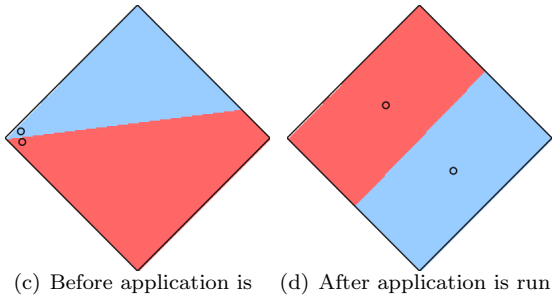
Figure 5: Exploration of Lloyd's on a regular 4-gon



(a) Before application is run

(b) After application is run

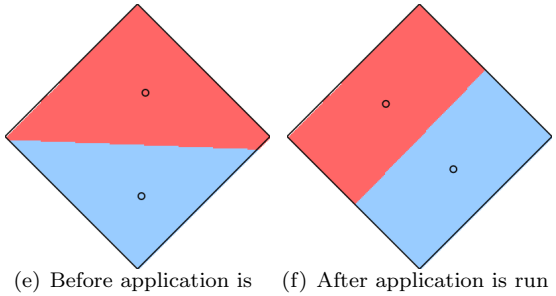
Next, we test when the points are placed side by side on of the vertexes.



(c) Before application is run

(d) After application is run

We now test when the points are places as close as possible to the diagonal CVT configuration.



(e) Before application is run

(f) After application is run

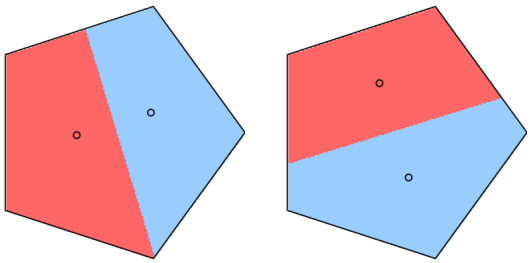
In Figure 5, we see that no initial placement of the two points results in the diagonal CVT configuration.

3.1.3 Regular 5-gon

We now construct parallel arguments using a pentagon, or regular 5-gon.

We know of at least two possible configurations of a 2-point CVT in a pentagon, seen below:

Figure 6: Known CVTs of a regular 5-gon

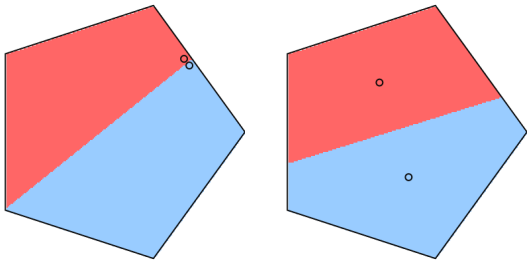


(a) Known CVT of a regular 5-gon across opposite vertexes

(b) Known CVT of a regular 5-gon across opposite edges

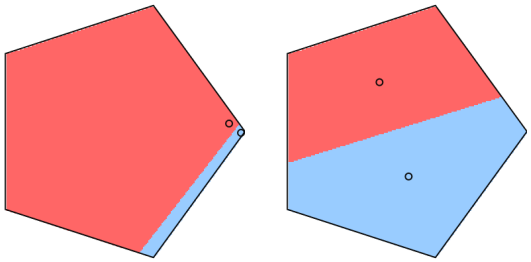
First, we will place the two points side by side on one of the edges.

Figure 7: Exploration of Lloyd's on a regular 5-gon



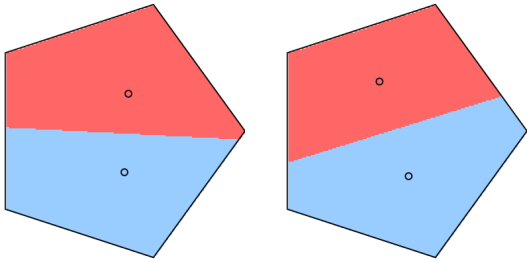
(a) Before application is run (b) After application is run

Next, we test when the points are placed side by side on of the vertexes.



(c) Before application is run (d) After application is run

We now test when the points are places as close as possible to the diagonal CVT configuration.



(e) Before application is run (f) After application is run

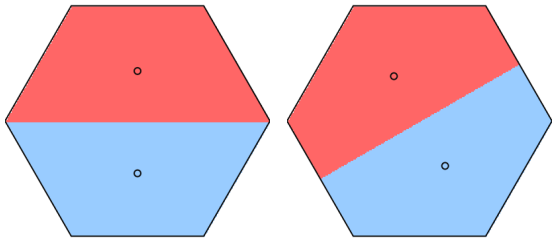
In Figure 7, we see that Lloyd's algorithm coverges to only Figure 7(b) for the regular 5-gon.

3.1.4 Regular 6-gon

We now construct parallel arguments using a hexagon, or regular 6-gon.

We know of at least two possible configurations of a 2-point CVT in a hexagon, seen below:

Figure 8: Known CVTs of a regular 6-gon

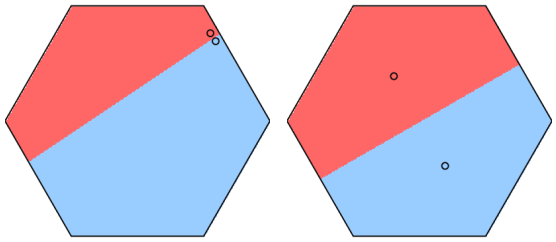


(a) Known CVT of a regular 6-gon across opposite vertexes

(b) Known CVT of a regular 6-gon across opposite edges

First, we will place the two points side by side on one of the edges.

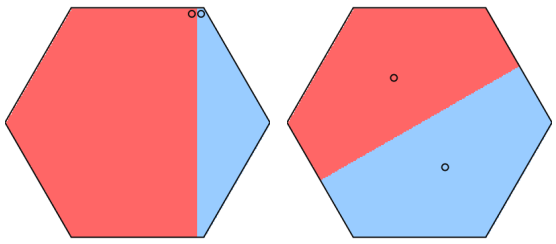
Figure 9: Exploration of Lloyd's on a regular 6-gon



(a) Before application is run

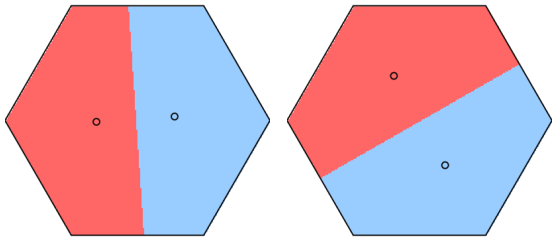
(b) After application is run

Next, we test when the points are placed side by side on of the vertexes.



(c) Before application is run (d) After application is run

We now test when the points are places as close as possible to the diagonal CVT configuration.



(e) Before application is run (f) After application is run

In Figure 9, we see that Lloyd's algorithm coverges to only Figure 9(b) for the regular 6-gon.

3.1.5 Stable CVTs of Lloyd's Algorithm

Based on exploration, in the context of Lloyd's Method, there seem to exist known CVT configurations that remain unrepresented by the applications we have created. The figures we have found we label as stable CVTs, in the context of Lloyd's method, such that the figure seems to be stably found and represented, alternative to the other known CVTs which are not found during these algorithm's convergence in this application. We thus conclude with the following conjecture:

Definition 3.1 (Stable CVTs)**.** *Lloyd's algorithm seems to prefer some configurations of CVTs over others, we will call these configurations stable CVTs in the context of Lloyd's algorithm.*

3.2 Characterizing Lloyd's Stable CVTs

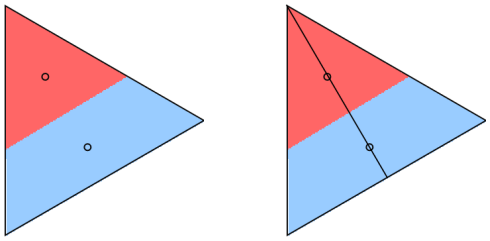
We now examine the configuration of the generator points within these stable CVTs in the context of Lloyd's method.

In a regular n-gon, if n is even, there exist two lines of symmetry in the figure, one from edge to edge and one from vertex to vertex. In this case, the edge to edge line of symmetry is shorter than the vertex to vertex line of symmetry. If n is odd, there is only one line of symmetry, from edge to vertex, of consistent length.

3.2.1 Lloyd's Stable CVTs and Lines of Symmetry

In the following figures, for each n-gon examined above, we show the stable CVT found above under exploration of the Lloyd's method application, as well as the known lines of symmetry in order to say something about the configuration of the stable CVTs.

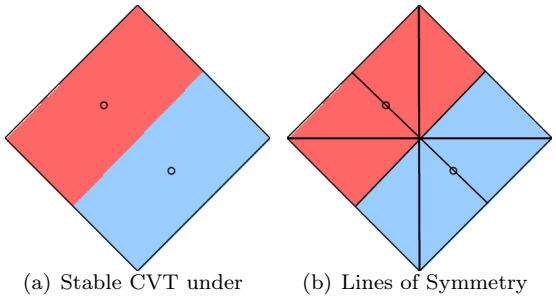
Figure 10: Stable CVT of a 3-gon



(a) Stable CVT under Lloyd's Method Application

(b) Lines of Symmetry

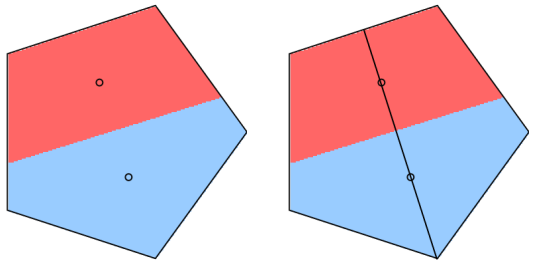
Figure 11: Stable CVT of a 4-gon



(a) Stable CVT under Lloyd's Method Application

(b) Lines of Symmetry

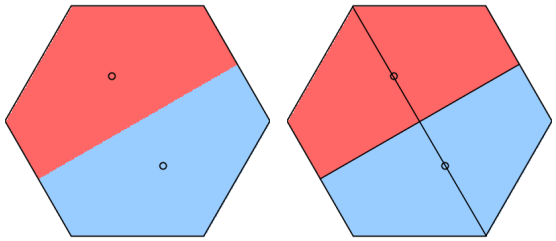
Figure 12: Stable CVT of a 5-gon



(a) Stable CVT under Lloyd's Method Application

(b) Lines of Symmetry

Figure 13: Stable CVT of a 6-gon



(a) Stable CVT under Lloyd's Method Application

(b) Lines of Symmetry

3.2.2 Generator Points in Lloyd's Stable CVTs

We now notice that in the 3 and 5 -gons, the generator points of these stable CVTs lie on the single line of symmetry through Ω.

In the the 6-gon, the generator points lie on the longest line of symmetry through Ω.

Finally, in the 6 -gon, the generator points of these stable CVTs lie on the shortest line of symmetry through Ω.

Thus we conclude with the following conjecture:

Conjecture 3.2. *Lloyd's algorithm, in this application, converges to a specific 2-point CVT representation, the stable CVT. In a regular n-gon,*

1. *if n is odd, the generator points lie on the single line of symmetry, from edge to vertex, through Ω.*

2. *if n is even and 4 does divide n, the generator points lie on the shortest line of symmetry, from edge to edge, through Ω.*

3. *if n is even and 4 does not divide n, the generator points lie on the longest line of symmetry, from vertex to vertex, through Ω.*

An alternative way of thinking about this would be,

Conjecture 3.3. *In the stable 2-point CVT, for Lloyd's algorithm on a regular n-gon, the boundary between the Voronoi regions lies on the shortest line of symmetry.*

4 Exploring Alternative Distance Metrics

We have now explored Lloyd's algorithm's convergence on 2-point CVTs. We will now introduce several new distance functions and examine how these formulas impact Lloyd's convergence and configurations.

4.1 Metrics

In mathematics, a metric or distance function is a function that defines a distance between each pair of elements of a set. In the above methods, as in most of mathematics, we use the standard Euclidean metric distance function in order to calculate which generator point a randomly generated point is closer to.

4.1.1 The Euclidean distance metric

First lets remember the equation for the Euclidean distance function and look at an example of how it would calculate distance. If we want to find the distance between (x_1, x_2) and (y_1, y_2), the equation we commonly remember is:

$$d(x, y) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2}.$$

Using this equation, if we want to calculate the distance between $x = (0, 0)$ and $y = (1, 2)$, we calculate the length of the diagonal line between the points.

$$d(x, y) = \sqrt{(1 - 0)^2 + (2 - 0)^2} = \sqrt{5}.$$

While most mathematics assume the Euclidean metric, there exist many other metrics which give interesting alternate results to algorithms, such as MacQueen and Lloyd's methods, that rely on a distance metric.

The discrete metric

One quick example of an alternative metric is the discrete metric, given by:

$$d(x, y) = 0 \text{ if } x = y \text{ and } d(x, y) = 1 \text{ otherwise.}$$

4.1.2 The Taxicab metric

We will now examine a less familiar metric, named the Manhattan distance, or Taxicab Metric. We will first introduce the metric then examine what is happening in this equation.

The Taxicab distance equation between (x_1, x_2) and (y_1, y_2) is:

$$d(x, y) = |y_1 - x_1| + |y_2 - x_2|.$$

This number is equal to the length of all paths connecting x and y along horizontal and vertical segments, without ever going back, like those described by a taxicab moving in a lattice-like street pattern, or grid to its destination.

Using this equation, if we want to calculate the distance between $x = (0, 0)$ and $y = (1, 2)$, we calculate the length of the horizontal and vertical line between the points. Note that any combination of horizontal and vertical lines will work.

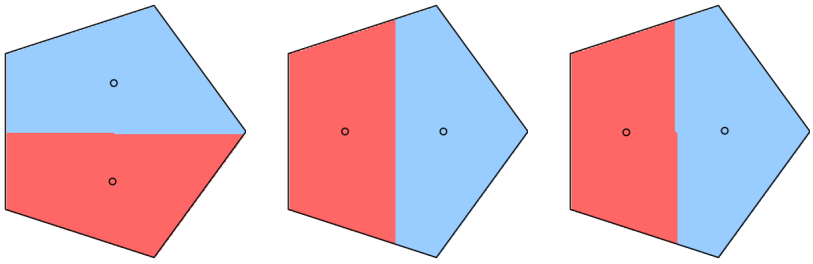
$$d(x, y) = |1 - 0| + |2 - 0| = 3$$

We see here that $\sqrt{5} \neq 3$, therefore these two equations find different values for distance. We will now introduce the taxicab distance equation to Lloyd's algorithm and explore the results.

Lloyd's With the Taxicab Metric

To begin, we create a new application at `carrot.whitman.edu/CVT/LloydsVariations/TaxiCabLloyds/` which implements Lloyd's algorithm using the Taxicab distance equation. We encourage continued exploration of this application, though currently no recognizable patterns of convergence or stability seem emergent. The following figure shows four different outcomes of the convergent figures of running this application on a 2-point 5-gon.

Figure 14: Taxicab Metric in Lloyd's Method on a 5-gon



This metric seems to reach multiple convergent configurations beyond those found with a Euclidean metric. The noticeable jagged line between the generator points is characteristic of the metric and may explain the lack of uniform convergence or stability.

Thus, we conclude that alternative distance metrics do effect the results of these conjectures and look to another metric to see if more interesting results can be found.

4.1.3 The Infinity Metric

In the past two metrics, we have remained in the L^p spaces, function spaces defined using a natural generalization of the p-norm for finite-dimensional vector spaces. They are sometimes called Lebesgue spaces. In $p = 1$, we get the Taxicab metric. In $p = 2$, we get the Euclidean metric. We now examine where p approaches infinity.

If we want to find the distance between (x_1, x_2) and (y_1, y_2), the equation is:

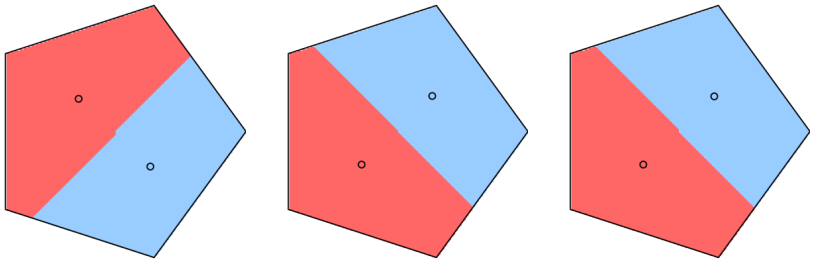
$$d(x, y) = \sqrt{(y_1 - x_1)^\infty + (y_2 - x_2)^\infty}.$$

Lloyd's With the Infinity Metric

This new metric is implemented at `carrot.whitman.edu/CVT/LloydsVariations/InfinityLloyds/`.

We again encourage continued exploration of this application, though currently no recognizable patterns of convergence or stability seem emergent. The following figure shows four different outcomes of the convergent figures of running this application on a 2-point 5-gon.

Figure 15: Infinity Metric in Lloyd's Method on a 5-gon



We conclude that as p approaches infinity, the application converges to seemingly similar configurations as under the Euclidean metric, with less precise and more variant results, as well as less defined lines surrounding each of the Voronoi regions. Though these metrics show alternate results on the convergence in these applications, it seems variant and less patterned.

5 Conclusion

In this paper, we examine the geometry of centroidal Voronoi tessellations. We show that while very little is known about centroidal Voronoi tessellations, using

p5.js as a platform to create visualizations provides us with additional tools to examine and conjecture upon their properties. These tools help us to successfully examine the generating algorithms and find Definition 3.1 and Conjecture 3.3 which further examine new properties of 2-point centroidal Voronoi tessellations.

References

[1] W. T. Blackburn, *Convex polygons and intersections of half-planes*, The Mathematical Gazette, 47 (1963), pp. 124–127.

[2] Q. Du and M. Emelianenko, *Acceleration schemes for computing centroidal voronoi tessellations*, Numerical Linear Algebra with Applications, 13 (2006), pp. 173–192.

[3] Q. Du, M. Emelianenko, and L. Ju, *Convergence of the lloyd algorithm for computing centroidal voronoi tessellations*, SIAM Journal on Numerical Analysis, 44 (2006), pp. 102–18.

[4] K. Gillespie, *2-point centroidal voronoi tessellations*, Whitman College Senior Project, (2016).

[5] V. F. Qiang Du and M. Gunzburger, *Centroidal voronoi tessellations: Applications and algorithms.*, SIAM Review., 41 (1999), pp. 637–676.