Separation of Voronoi Areas

Martin Gebert, Sascha Schreckenbach, Jonathan Sharyari

28th January 2013

Abstract

Given a set $\mathcal{R}$ of N red points, we consider a set $\mathcal{B}$ of blue points to be a *solution*, if in the Voronoi diagram for $\mathcal{R} \cup \mathcal{B}$, no two red points have coinciding delimiters. In this paper we experimentally investigate the claim that an minimal solution always needs as many blue points as there are red points. Although this claim has not been proven, we have found no counter-examples to this claim, further strengthening our belief that the claim is in fact true.

Contents

1 Introduction

1.1 Voronoi Diagram

Given a set of points in space, a Voronoi diagram is a partitioning of the space into a set of Voronoi regions or ”cells”. The Voronoi region corresponding to a point p consists of all points that are closer to p than to any other point in space.

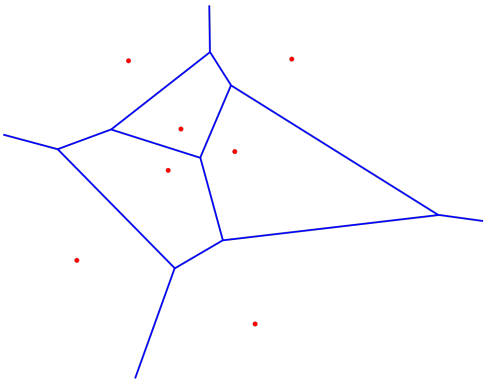

Figure 1: A Voronoi diagram.

1.2 Delaunay triangulation

A Delaunay triangulation for a set of points in space is a triangulation of the points, with the added property that no point is inside the circumcircle of any triangle in the triangulation.
The Delaunay triangulation is dual to the Voronoi diagram.

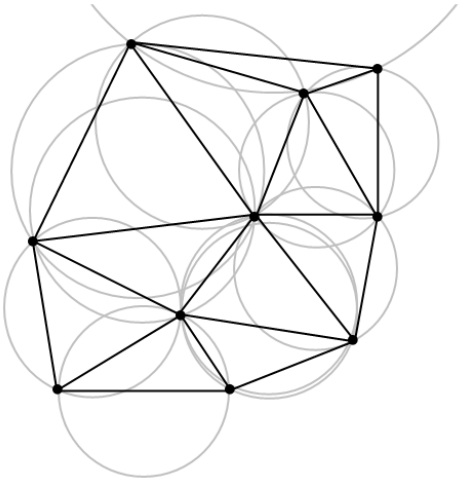

Figure 2: A Delaunay triangulation, with a set of corresponding Delaunay circles.

1.3 Arrangement

Given a set of planar curves, the arrangement is the subdivision of the plane into zero-dimensional, one-dimensional and two-dimensional cells, called vertices, edges and faces, respectively induced by the given curves. 1

1.4 The Set Cover Problem

Given a set $\mathcal{A}$ of sets of numbers, the set cover problem is the problem of finding a minimal subset $\mathcal{B}$ of $\mathcal{A}$, so that the union of the sets in $\mathcal{B}$ is equal to the union of all elements of $\mathcal{A}$.

1.4.1 Example

Let $\mathcal{A} = \{\{1,2\}, \{2,3\}, \{2,5\}, \{3,5\}\}$. The union of all sets of $\mathcal{A}$ is $\{1,2,3,5\}$.
The subset $\mathcal{B}_1 = \{\{1,2\}, \{2,3\}\}$ has the union $\{1,2,3\}$ and does not cover the set $\mathcal{A}$.
The subset $\mathcal{B}_2 = \{\{1,2\}, \{2,3\}, \{2,5\}\}$ has the union $\{1,2,3,5\}$ that covers $\mathcal{A}$, but it is not minimal.
A minimal solution is $\mathcal{B}_3 = \{\{1,2\}, \{3,5\}\}$.
The set cover problem is known to be NP-complete.4

2 Problem formulation

Given a set $\mathcal{R}$ of red points, find a set $\mathcal{B}$ of blue points such that in the Voronoi diagram of $\mathcal{R} \cup \mathcal{B}$, no two regions corresponding to points in $\mathcal{R}$ are incident to each other. For the Delaunay triangulation of $\mathcal{R} \cup \mathcal{B}$, this means that there is no edge connecting two red points. The conjecture that $|\mathcal{B}| = |\mathcal{R}|$ is sufficient in an optimal solution is to be experimentally explored.

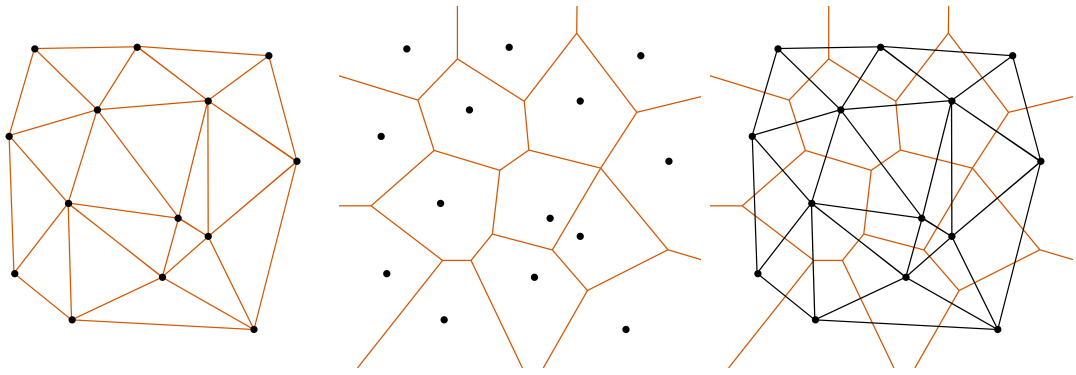

Figure 3: A triangulation of a set of points (left), the corresponding Voronoi diagram (middle) and the overlapping of the triangulation and the Voronoi diagram, emphasizing the dual relationship (right).

3 Background

3.1 Upper and lower bounds

Given a set $\mathcal{R}$ of red points with $N = |\mathcal{R}|$, it has been shown 2 that at least N-1 points are needed to solve this problem. This is exemplified in the picture below, and always applies to the special case when all points lie in row. In the general case, it has been shown that for $N > 2$ there are cases where at least N blue points are required to solve the problem.

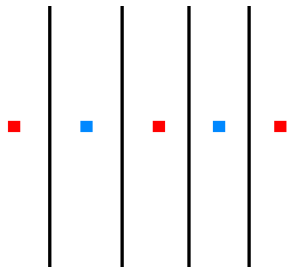

Figure 4: Example of points not in general position, having a solution with N-1 blue points.

In the general case, it has been proven 2 that at most $3N/2$ blue points are needed to solve the problem, and in the case where the points in $\mathcal{R}$ are in convex position, $5N/2$ points are sufficient. It has been conjectured that N points are sufficient to solve the problem in the general case.

3.2 Complexity

Although not yet shown, the problem is suspected to be NP-hard 2, as a similar problem has been proven to be NP-hard. 3

4 Algorithms

4.1 General outline

1. For the set $\mathcal{P}$ of points (both red and blue) find the Delaunay triangulation and the Voronoi diagram.

2. For each edge in the Delaunay triangulation that connects two red points, find a circle with these two points on its perimeter and the circle's centre on the corresponding Voronoi edge and add these to the (initially empty) set of circles $\mathcal{C}$.

3. Calculate the arrangement $\mathcal{A}$ of the circles $\mathcal{C}$.

4. Find a set of points $\mathcal{P}^*$, so that for every face in the arrangement $\mathcal{A}$, exactly one point is located in the interior. For every point $\rho \in \mathcal{P}^*$ and every circle ς in $\mathcal{C}$, determine whether ρ is in the interior of ς.

5. Find a minimum subset $\mathcal{P}^{**}$ of $\mathcal{P}^{*}$, so that every circle in $\mathcal{C}$ is covered by at least one point.

6. Set $\mathcal{P}$ to $\mathcal{P}^{**} \cup Red(\mathcal{P})$, where $Red(\mathcal{P})$ denotes the set of red points in $\mathcal{P}$. Calculate the Delaunay triangulation for $\mathcal{P}$.

7. For every red edge still in the triangulation, calculate the length of its dual in the Voronoi graph. If no red edges exist in the triangulation, terminate. If the sum of the calculated lengths is sufficiently small, use random search to find an optimal solution. Otherwise, restart from 2.

4.2 Finding circles corresponding to unsatisfied edges

Following the terminology used by the CGAL-project, a **line** is a line unbounded on both sides, whereas an line that is bounded on one side is called a **ray** and when bounded on both sides it is called a **segment**.
The approach for choosing a circle depends on which type of Voronoi edge is to be blocked. Given a Voronoi edge corresponding to the neighbouring points p_1 and p_2, there are the following cases:

1. For a **line**, the smallest possible circle is chosen. This is the circle with its centre in the middle of p_1 and p_2, and a radius such that p_1 and p_2 are on the perimeter of the circle. A line only occurs in the Voronoi diagram in the trivial case where all points lie in a straight line.

2. A **ray** in the Voronoi diagram is always associated with two points on the convex hull of the Delaunay triangulation. There are arbitrarily large circles with the points p_1 and p_2 on their perimeter and their centre on the ray. In this case, one of these circles is randomly chosen.

3. For a **segment**, the circle with p_1 and p_2 on its perimeter, and its centre on the midpoint of the segment is chosen.

4.3 Finding a point in the interior of a face

A relatively difficult task proven to be that of finding a point in the interior of an arbitrary face, given two point p_1 and p_2 known to be on the boundary of the face. The following algorithm was used.

- Find the point p_m in the middle of p_1 and p_2.

- Find the line l that is going through p_m and is perpendicular to the edge between p_1 and p_2.

- Find the segment s that is the intersection of l and the smallest rectangle large enough to contain all circles of the arrangement. This step is needed because the the kind of CGAL arrangement used does not support lines. This step is described here only for the sake of completeness.

- Find the segment that is the intersection of the segment s and the face itself.
- Any point except the endpoints of this segment are inside the face. The midpoint of the segment is chosen.

4.4 Finding a minimal subset of points

The problem of finding a minimal subset of a set of points $\mathcal{P}^*$, so that at least one point in $\mathcal{P}$ is in the interior of each circle in $\mathcal{C}$, is exactly that of the set covering problem. The problem is easily stated in terms of an integer programming problem:

For every point $\rho_i \in \mathcal{P}^*$, let $x(\rho_i)$ be a boolean, set to 1 if the point ρ_i appears in the currently considered solution and 0 otherwise. Also, for every circle $\varsigma_j \in \mathcal{C}$, let $\varsigma_j(\rho_i)$ be 1 if point ρ_i lies in the interior of c_j and 0 otherwise.

1 Minimize: $\sum_{\rho \in \mathcal{P}^*} x(\rho_i)$

2 Constraint: $\sum_{\rho \in \mathcal{P}^*} \varsigma_j(\rho_i) \geq 1$ **for** each $\varsigma \in \mathcal{C}$

Stated in this way, the problem can be solved relatively efficiently with an integer programming solver, such as Gurobi.

4.5 Refining a near-optimum solution through random search

Random search is a direct search method that does not require the derivative of the function to be minimized. It is algorithmically simple, and in general terms work as follows:

To minimize a function $f : \mathbb{R}^n \to \mathbb{R}$, start with a (potentially random) solution $x : \mathbb{R}^n$ and do the following:

```
1  while (f(x) > 0 )
2    Generate a new candidate solution y in the neighbourhood of x
3    if f(y) < f(x)
4      replace the current solution x with y.
```

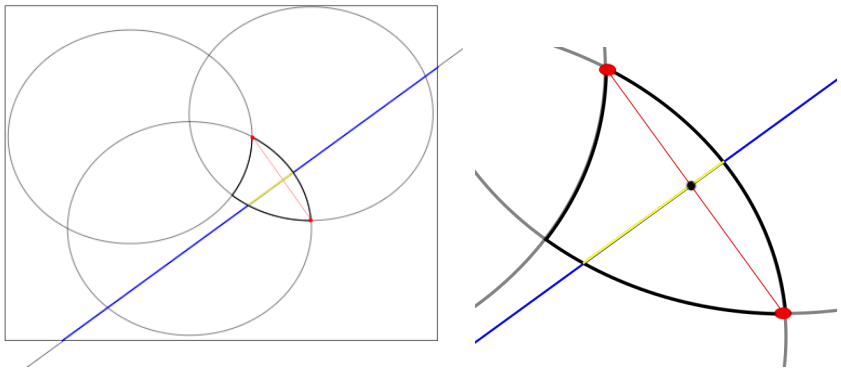

Figure 5: The graphic shows the application of the algorithm descibed in this section to an exemplary face.

There are several candidates for choosing the objective function f, for example the number of red Voronoi edges. For this problem, we have chosen the sum of the squared lengths of all red voronoi edges.

5 Results

5.1 Graphical interface

In order to easily specify new problems, and to visualize the process of solving the problems, a minimalistic graphical interface is used. It has methods for adding, moving and removing points. It automatically updates and draws the Voronoi diagram corresponding to the set of points used in the current iteration of the above mentioned algorithm, and also the set of Delaunay circles corresponding to the red edges in this Voronoi diagram.

Additionally, methods for saving and loading configurations were added.

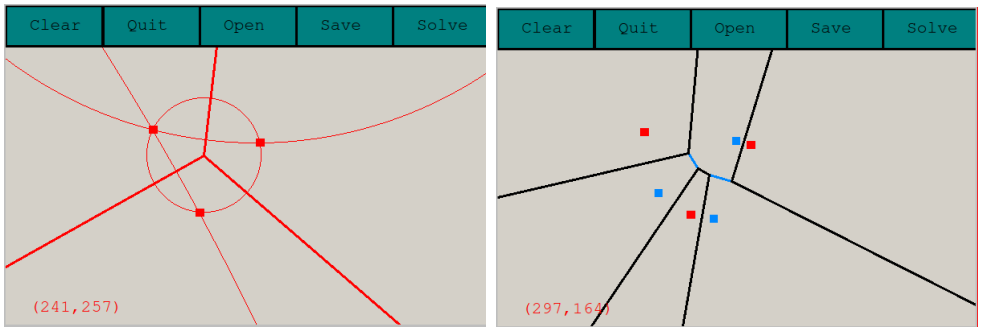

Figure 6: An example of a problem configuration (left) and a found solution to the same problem (right).

5.2 Command line parameters

It's also possible to start the program without the GUI. To do so, either the name of an existing file or "random" followed by an integer has to be provided as command line parameter. If the first command line parameter is "random", no GUI is displayed, and a random problem, that is, a set containing the provided number of randomly placed red points, is generated and solved. Afterwards, the number of red points in the problem, as well as the number of blue points used to solve the generated problem is displayed along with the runtime of the program in milliseconds:

```
user@computer: /path/$ ./separator random 7
7, 7, 4153
user@computer: /path/$
```

If a filename is provided, the problem saved in the named file is solved (without GUI) and the same information as above is printed.

5.3 Automated testing and debugging

The GUI-less mode was designed to be used for automatation, such as the provided by the `randomtest` and `reruntes`" scripts.

The `randomtest` script will randomized problems of a specified size, a certain number of times (this is specified in the script). If a problem could not be solved within 5 minutes, it is considered to have timed out, and is stored in the subfolder *testing*.

The script `reruntest` will run all tests in the *testing* subfolder again, and remove them after completion. If a test again times out, a new file will be created in the same folder. This is to ensure that a correct solution is found for all problems.

When running with GUI enabled, the last used configuration is always solved in the file *latest.cnfg*. This means that in case of unexpected behavior, for example a crash, the configuration on which it occured can always be recovered and examined.

5.4 Speed

The problem of finding a blocking set of blue points is computationally hard, since the set-cover problem is NP-hard and also by its own, assuming the conjuncture in 2 is true. This complexity is reflected in the time it takes to solve a problem, shown in figure 4.

The figure shows the average time of solving a problem with between 7 and 12 points. The program was run on forty randomly generated problems for each problem size. Due to the existence of local minima in the search space explored by the random search algorithm, there is a small risk that a run takes unusually long time. Such statistical *outliers* greatly impact the total average, and thus both a graph with and without these outlier values have been included.

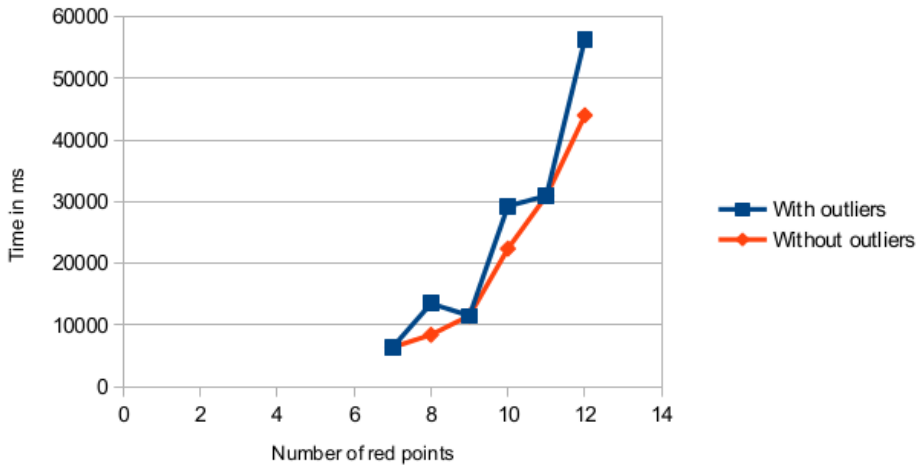

Figure 7: Graph showing average time to find a solution, for problems of different size. The average is based on forty runs, and the results are shown both with and without outlier values.

In all the tested problems, the number of blue points in the final solution was the same as the number of red points in the original problem.

5.5 Correctness of the calculated solutions

5.5.1 Inexact calculations

The numbers used in the algorithm implementation are mostly floating point numbers. While calculations involving floating point numbers are usually not totally precise, the CGAL library offers the possibility to perform 100% exact calculations, although with the disadvantage of a notable performance decrease. Thus, only step 2 of the algorithm is done with full precision.

It is theoretically possible that a set of blue points is found, that is believed to covers all circles, when it in reality does not (or vice versa). Note that the Delaunay triangulation that is later constructed using the found points does not depend on the calculated circles and their precision. It is when this Delaunay triangulation is free from red-red neighbours that the program terminates, and thus the correctness of the final solution is not affected by the inexactness of the intermediate calculation, although it is possible that more iterations are performed than what is necessary.

The solution returned are not necessarily correct, since the construction of the Delaunay triangulation also uses inexact calculations. It is therefore possible that a solution returned by our program is in fact a false solution, or that a true solution is overlooked. This may lead to a solution using more blue points than necessary will be accepted and returned. The only way to guarantee that the returned solutions are always correct and minimal would be to do all calculations in an exact way. As exact calculations are costly, it is preferable to perform inexact calculations and in the rare cases where a non-optimal solution is found, try solving the problem again from the beginning.

5.5.2 Finding a point in the interior of a face

Another design issue was that of finding a point in every face of the arrangement (step 4 in section 4.1). The algorithm presented in section 4.3 provides satisfying results, but is relatively complex.

An alternative approach is to simply choose two points on the boundary of a face, and chose the middle point as result, without checking whether it really lies in the considered face. In fact, such a point will be within the face 50% of the cases, since the boundary of the face in a restricted area is convex or concave with the same probability.

This latter approach is computationally easy, but does not fully fulfill the requirement of the algorithm. As explained above, it can not lead to incorrect solutions being returned by our program, but it might happen that the returned solution is not minimal. Again, this is a rare phenomenon and the performance gain of using the inexact method is large, and therefore this approach is preferable. The methods were both implemented in a manner such that switching between them is easy.

The former approach is slower to the extent that a comparison in speed is not possible, other than for very small problems (up to five red points). For bigger problems, a large proportion of tests time out making automated testing cumbersome.

6 Conclusions

The purpose of this project has been to experimentally explore the claim, that in an optimal solution there are as many blue points as red points, except for the case where all red points lie in a row. It must be noted that this can be falsified through experimentation by finding a counter-example to the claim, but it can never be experimentally proven. During testing, no counter-examples to the claim have been found, leaving us to believe that this conjecture is in fact true.

7 Discussion

During the course of this project, we have used our tool to solve a large amount of problems, but with the big limitation with respect to performance. Due to the exponential complexity of this problem (shown in figure 7), we have been limited to problems of a relatively small size. To more fully investigate this claim, the solver would likely need to be improved.

One way to do this is to more fully investigate how and when random search is effective. During our testing, a random search stage was entered when the total length of the red segments in the voronoi diagram was smaller than 300*N, where N is the number of red points. This is a very rough estimation, which if too large leads to a local minima, and if too small means the problem could be solved faster. A second parameter, the amount of randomization used, could also be optimized. It is likely that using a similar but more sophisticated technique, such as tabu search, would be more effective.

Another way to improve our solver would be to add more fast heuristics that are likely to find a solution, so that the slow but (almost) exact algorithm described in section 4 would be used rarely.

8 Tools

In this project, a set of tools and libraries were used. Most notably, the Computational Geometry Algorithms Library (CGAL) provides a vast number of efficient algorithms for different areas within computational geometry. In this project, we have made extensive use of the methods for calculating Voronoi diagrams, triangulations and arrangements.

The Gurobi optimzer was used for solving the integer programming problem. It provides an easy-to-use C++-API.

To easily build the simple GUI for our program, we used QT.

Finally, we used git for revision control and to simplify our collaboration. The source code for the project is available at `https://github.com/martin-mfg/voronoi-thesis-tester`

9 references

1. Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin
   CGAL manual chapter 20, 2D Arrangements
   http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/Arrangement_2/Chapter_main.html

2. O. Aichholzer, R. Fabila-Monroy, T. Hackl, M. van Kreveld, A. Pilz, P. Ramos, und B. Vogtenhuber

   Blocking Delaunay Triangulations.

   Proc. Canadian Conference on Computational Geometry, CCCG 2010, Winnipeg, August 9/11, 2010.

3. M. de Berg, D. Gerrits, A. Khosravi, I. Rutter, C. Tsirogiannis and A. Wolff.

   How Alexander the Great Brought the Greeks Together while Inflicting Minimal Damage to the Barbarians.

   Proc. 26th European Workshop on Computational Geometry, pages 73-76, 2010.

4. Richard M. Karp Reducibility Among Combinatorial Problems. In: R. E. Miller, J. W. Thatcher (Hrsg.): Complexity of Computer Computations. Plenum Press, New York 1972, S. 85-103.

9.1 images

- Delaunay circumcircles, GNU Free Documenation Licence, Nü es

  http://commons.wikimedia.org/wiki/File:Delaunay_circumcircles.png

- Delaunay triangulation picture 1-3, public domain, user Capheiden

  http://upload.wikimedia.org/wikipedia/de/1/17/Voronoi-Delaunay.svg

  http://upload.wikimedia.org/wikipedia/de/4/48/Voronoi-Diagramm.svg

  http://upload.wikimedia.org/wikipedia/commons/1/1f/Delaunay-Triangulation.svg