University of Wisconsin Milwaukee

UWM Digital Commons

Theses and Dissertations

May 2020

Non-Discriminatory Service Robot Placement Using Geometric Median

Brian Boyd
University of Wisconsin-Milwaukee

Follow this and additional works at: https://dc.uwm.edu/etd

Part of the Computer Sciences Commons

Recommended Citation
Boyd, Brian, "Non-Discriminatory Service Robot Placement Using Geometric Median" (2020). *Theses and Dissertations*. 2356.
https://dc.uwm.edu/etd/2356

This Thesis is brought to you for free and open access by UWM Digital Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of UWM Digital Commons. For more information, please contact open-access@uwm.edu.

NON-DISCRIMINATORY SERVICE ROBOT PLACEMENT USING GEOMETRIC MEDIAN

by

Brian Boyd

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin-Milwaukee

May 2020

ABSTRACT

NON-DISCRIMINATORY SERVICE ROBOT PLACEMENT USING GEOMETRIC MEDIAN

by

Brian Boyd

The University of Wisconsin-Milwaukee, 2020
Under the Supervision of Professor Amol Mali

Service robots are becoming increasingly common, and businesses are adopting their use at an increasingly rapid rate in order to reduce costs and provide efficiencies in performing mundane tasks. However, very little research has been performed in order to understand and address ethical concerns regarding their deployment and use.

One such concern is how one can ensure placement of a service robot such that is does not discriminate either in favor of or against individuals. This research explores techniques that can be used to provide a quantitative methodology to ensure fairness in terms of service robot placement such that discrimination does not occur.

These techniques include the development and further enhancement of a heuristic hill climbing algorithm used to approximate the Geometric Median (GM). This algorithm is then benchmarked against Weiszfeld's Algorithm, a well-known algorithm commonly used to solve the GM problem.

These two algorithms are then visualized using Dynamics Explorer, an open source software tool, to create 2d maps of the dynamics of their convergence rates along with maps of

F(), the “sum of the Euclidean distances” function underlying the calculations used by both GM approximation algorithms.

The heuristic hill climbing algorithm is also extended to handle obstacles being introduced into the service robot’s workspace.

It is further shown that as the size of ξ approaches ∞^+, the Geometric Median converges to the centroid, given certain assumptions, such as the target points being evenly distributed in the plane.

© Copyright by Brian Boyd, 2020
All Rights Reserved

To

my wife,

my children,

and all of mankind.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Statement of the Problem

Robots are becoming increasingly common in our daily lives, largely due to rapidly improving technologies like machine learning and artificial intelligence, big data and business informatics, and the adoption of cloud platforms. These technologies, coupled with business objectives such as the desire for increased productivity and reduction in payroll costs, are driving the adoption of robots in novel new ways.

Some of these robots are on the front line, servicing humans directly, and they are known collectively as service robots. Service robots come in many shapes and forms, some of which you may be familiar with, such as robotic vacuum cleaners, or robotic merchandise ordering kiosks at some retail or stores or restaurants.

From a business perspective, the motivating force behind the deployment of service robots has been cost reduction, since it is cheaper to buy a robot than to utilize a human for many of these mundane, rote activities. However, it appears that very little consideration is being given to the overall impact these robots have on society, and how their very presence or absence can impact humans on a psychological level.

This paper attempts to address a single aspect of that picture: given a place that people use for some purpose, such as a restaurant or shopping center, is there a way that we can deploy a service robot to assist those people such that it is placed in a location that quantitatively minimizes the perception that it discriminates for or against any of those people by virtue of being placed closer or further away from some people rather than others? To put it another way, can we mathematically determine a place to put this robot that is “fair”?

Non-discriminatory placement describes the desire to place a service robot in a location such that the potential customers of the service are unlikely to feel discriminated against. Examination of the problem resulted in the understanding that the underlying problem is closely related to the problem of facility location in many ways. Thus, the GM was chosen as it minimizes the amount of travel required for a set of people to access the services offered by the robot, providing an objectively fair way to reduce perceived discrimination. By using the GM, one can show that a given service robot is not intentionally placed closer or farther from any individuals seeking the services being offered. Techniques for ensuring non-discriminatory placement of service robots are developed by employing algorithms used to approximate the GM. To this end, a heuristic algorithm was developed which approximates the GM within a specified epsilon (ε) bound. This algorithm was analyzed and benchmarked against a well-known algorithm commonly used to approximate the GM, known as Weiszfeld's Algorithm. Both algorithms use iterative methods to achieve their goals, however the heuristic algorithm described in this thesis is a modified hill-climbing algorithm, while Weiszfeld's is a type of iteratively re-weighted least squares algorithm.

The first case which was examined is that of an arbitrary number of people occupying various locations within a 2d plane representing a place people gather to get access to some service. Within this place there is a robot providing a service to the people present, and the goal is to find a location for the robot such that the people desiring the service are not likely to feel discriminated against.

The second case which was examined added a constraint in the form of arbitrary polygonal shapes being introduced into the place which represent furniture or other obstacles which preclude parts of the plane from being used. In this case, the heuristic algorithm was extended to find locations near the GM but located within the free space, clear of any obstacles. Other algorithms referenced in this paper do not account for these obstacles.

The third case which was examined inverted the use of the polygons in the second case such that they now represented the free space, and all other space was unavailable for use by the robot. In this third case the heuristic algorithm was extended to locate points within each polygonal free space close to the GM as potential sites for placement of the robot.

Lastly, modifications were made to an existing open source software project in order to visualize some aspects of this research, including the construction of 2d maps displaying the convergence rate for each algorithm on an entire plane, the construction of a 2d heat map representing the each point's average distance to the set of people provided, and interactive real-time tools to watch the algorithms perform their calculations based on where the user clicks on the plane.

Results show that in general, the heuristic algorithm's performance is lower than that of Weiszfeld's, however it never fails to converge. Weiszfeld's required modifications to ensure it didn't get "stuck" when an iterate got too close to one of the target points. Analysis of the data output by the algorithms shows that if the target points are randomly scattered throughout the plane, as the number of points approaches ∞, the GM converges to the centroid.

1.2 Definition of Terms

Centroid point in a plane representing the arithmetic mean position of a set of points. Formulaically, the centroid = $(X_1 + \ldots + X_k) / k$ [17], for k points X_1 to X_k in $\mathbf{R}^n$.

Epsilon (ε) double value, user configurable, describes the minimum bound at which an algorithm terminates because it is approaching a fixed point due to the current iterate being less than ε away from the last iterate in Euclidean space

F() value the calculation of the sum of the Euclidean distances from a given point to all members of ξ

Geometric Median (GM) given a discrete set of sample points in a Euclidean space, this is the point minimizing the sum of the distances to those sample points

Starting point the initial starting point for an algorithm, may be arbitrary or user defined

Target point set (ξ) in terms of the geometric median problem, this is the set of sample points being used to calculate the GM

1.3 Description of the Remaining Chapters

In Chapter 2, we perform a review of existing literature, describe how the problem is an analogous to the facility location problem, and describe the Geometric Median.

In Chapter 3, we describe the algorithms we developed and discuss how they work. We then describe the extensions made to the heuristic algorithm to account for the additional test cases involving polygonal obstacles and free space. We finish the chapter with a discussion of the extensions made to Dynamics Explorer allowing us to visualize the dynamical behavior of the two approximation algorithms.

In Chapter 4, we describe the results of our experiments, including measuring their absolute and relative performance. We also discuss factors affecting the convergence rates for each algorithm and conclude the chapter by describing the import of the visualizations produced.

In Chapter 5, we discuss our conclusions and provide some ideas for further research and work in this area.

2. Background and Related Work

2.1 Review of Existing Literature

Very little literature exists regarding the intersection of ethical concerns, such as discrimination, and robots. However, the paper by Wirtz, et al [1] does touch on many ethical concerns from a macro level and points out that there are many valid concerns that leaders and researchers should be keeping in mind as the frontier of advancement begins to include more service robots in front-line roles interacting with humans.

There is a growing library of research in the field of service robots, and much of it is relatively recent. Park, Yu, and Cho [4] studied the increased effectiveness of service robots offering customized service to customers rather than a generic one. Kobayashi, et al [5] examines the development of a service robot designed to serve tea to elderly patients, and feelings of dissatisfaction among patients due to the behavior of the robot in failing to provide feedback to those waiting for service while it provides service to another individual. Stock and Merkle [6] perform a qualitative study of robot – human interactions designed to examine human user's acceptance of robots offering services and how the expectations on robotic servers differs from human servers. Samarakoon, et al [7] examine some micro-level parameters used to determine how a service robot should approach a potential human user, including examining the human's behavior to determine the appropriate proxemics given the context of their interaction.

The geometric median is an old problem which has been studied extensively, and there exists a robust library of resources describing methods used to analyze the problem and approximate the solution. In the context of computer science, the first real breakthrough came with the introduction of an iterative algorithm formulated by E. Weiszfeld in 1937[2]. This algorithm went largely unnoticed for a number of years but was eventually rediscovered and analyzed by Kuhn in 1973[3]. Belas and Yu [8] in turn examined Kuhn's work and provided an alteration to the algorithm which removed the possibility for the algorithm to get "stuck" if an iterate got too close to one of the members of ξ. More recently, Aftab, Hartley, and Trumpf [9] describe a generalized version of Weiszfeld's Algorithm which can be used to find the L_q solution for $1 \leq q < 2$ in order to solve a wider variety of problems. The L_q solution minimizes the sum of the qth power of errors. Weiszfeld's being the classic example of an algorithm for finding the L_1 solution. There are also several papers discussing the use of approximations of the GM having applications in artificial intelligence and big data: [10], [11], [12], [13], and [14].

2.2 Facility Location as an Analogue

When reduced to a conceptual form, it became apparent that the problem is a direct analogue to the facility location problem. That is, consider a plane which contains a set of target points. That set of points represents locations that need to be visited, serviced from some single focal location. We are searching for the point in the plane that minimizes the sum of the distances from the focal location to the set of target points. That minimum point is the GM.

2.3 Overview of the Geometric Median

The Geometric Median is the point in some Euclidean space which minimizes the sum of the distances to a set of target points. When presented as a problem involving three non-colinear points in a plane, it is known as Fermat's problem, and has a solution which produces an exact answer, known as the Fermat point.

For the case of 4 co-planar points, there are two cases to consider:

1. One of the points is inside a triangle formed by the other three points. In this case, that interior point is the Geometric Median.
2. Otherwise, the four points form a convex quadrilateral, and one can simply calculate the crossing point of the diagonals to find the Geometric Median.

The solution to case 2 was first provided by Giovani Fagnano, a theologian and mathematician, in the mid 1700's. Fagnano did not consider case 1, however the solution was provided later by Johann Radon in 1921 in a paper titled "Mengen konvexer Körper, die einen gemeinsamen Punkt enthalten", in which he developed the Radon theorem on convex sets [15].

For 5 or more, non-colinear points, there is no known exact formula to calculate the Geometric Median. In fact, Bajaj proved in 1986 that the GM problem in non-solvable [16]. Rather, approximation techniques must be used to determine a point within some ε that satisfies the needs of the person seeking the solution.

Kuhn [2] showed that Weiszfeld's Algorithm converges globally, and under some conditions, in linear time. Cohen, et al [14] discuss a variety of GM approximation techniques representing the fastest algorithms, in terms of asymptotic time, with performance envelopes guaranteed to be at worst polynomial in time. They also present new algorithms which obtain nearly linear time performance using very advanced interior point methods.

3. Methods Used

3.1 Overview of algorithms developed

Several algorithms were developed and then implemented in the Java programming language. The implementations were performed by the same author, ensuring that the level of sophistication employed played no factor in the results. The first algorithm is used as a control, and is a modified version of Weiszfeld's algorithm, with additional logic to ensure that if an iterate gets too close to a target point, the algorithm does not become "stuck". The second algorithm is an experimental heuristic algorithm based on a hill climbing methodology. The third algorithm utilizes the heuristic algorithm, however, rather than finding an approximation for the geometric median, it attempts to minimize the difference between the average distance from a given test point to two sets of target points representing people belonging to different categorizations of a discrimination category (such as male or female.)

3.2 Weiszfeld's algorithm – the control

Weiszfeld's algorithm is a commonly used approach to approximating the GM, and so it is used as a control. Originally described by Endre Weiszfeld in the Tohoku Mathematical Journal in 1937, this algorithm went widely unnoticed for many years before being rediscovered and subsequently analyzed by Harold Kuhn in a paper published in the journal Mathematical Programming in 1973.

The algorithm is a type of iteratively re-weighted least squares in which an arbitrary initial starting point is chosen, and over successive iterations improved approximations are generated based on the results of each previous iterate. One may iterate the algorithm until it reaches a fixed point given some ε.

The following formula describes an arbitrary iterate in this scheme:

$$y_{i+1} = \left(\sum_{j=1}^{m} \frac{x_j}{\|x_j - y_i\|} \right) \Big/ \left(\sum_{j=1}^{m} \frac{1}{\|x_j - y_i\|} \right).$$

- y_i is the current iterate, or "best guess" approximation
- y_{i+1} is the next iterate for the "best guess" approximation
- x_j is one of the '*m*' sample points in ξ

As Kuhn noted, the algorithm can fail to converge when an iterate gets too close to a point contained in the set of target points being compared against. In this situation, the algorithm gets "stuck" and treats the point as a fixed point, even though it is not the global minimum. To correct for this, our implementation includes additional logic to detect the situation and find a suitable alternate iterate. This method was described by Belas and Yu (1982) [8].

Pseudo-Code for Weiszfeld's Algorithm

```
ε ← <some configured value, very small, positive>
ξ ← target point set
diff ← $\infty^+$
P ← starting point
sum_w ← 0
$Q_k$ ← new point, (0, 0)
While diff > ε
    for each point in ξ
```

```
        Q ← 'this' point in $\xi$
        if P == Q, execute subroutine to break out of errant minimum
        else, continue
        d ← dist(Q, P)              <get Euclidean distance>
        w ← 1/d                     <calculate this weight>
        sum_w ← sum_w + w           <accumulate sum_q>
        $Q_k$ ← $Q_k$ + (Q * w)     <vector addition, multiplication of Q's values>
    $Q_k$ ← $Q_k$ / sum_w
    diff ← dist($Q_k$, P)
    P ← $Q_k$
end While
```

3.3 Heuristic algorithm – the experiment

This algorithm is a type of hill climbing algorithm which iterates until it reaches a fixed point given some ε. From an initial starting point as the current iterate, search in a circle using polar coordinates to locate a candidate point at STEP_SIZE distance with a lower F() value than the current iterate. Move in the direction of the vector described by the line segment connecting the current iterate and the candidate in STEP_SIZE increments, testing the candidate point at each step, and continuing in that direction as long as the F() value of new candidates is smaller than the previous candidate. Once we reach a candidate point which no longer has a lower F() value, we reduce STEP_SIZE and do another circular search.

In this fashion, the algorithm "flows downhill" until it reaches a minimum, given some configured ε.

Pseudo-Code for Heuristic Algorithm

```
Centroid ← calculate centroid using known formula
ClosestPoint ← Centroid                                  <initial starting point>
ClosestPointF ← F(ClosestPoint)
CheckedPoints ← add ClosestPoint to hashset
STEP_DELTA ← CONFIGURED BY USER                          <value between 0 and 1>
STEP_SIZE ← CONFIGURED BY USER
OuterLoop:
While STEP_SIZE > ε
    iteration++
    NewClosestPoint ← TRUE
    While NewClosestPoint == TRUE
        NewClosestPoint ← FALSE
        Neighbor ← GetNextPoint()
        NeighborF ← F(Neighbor)
        If NeighborF < ClosestPointF
            NewClosestPoint ← TRUE
            diff = dist(ClosestPoint, Neighbor)
            ClosestPoint ← Neighbor
            ClosestPointF ← NeighborF
            CheckedPoints ← add ClosestPoint to hashset
            movecount++
            if diff < ε
                break OuterLoop
    end While
    STEPSIZE ← STEP_SIZE * STEP_DELTA
end While
```

3.4 Heuristic Algorithm Extension – Polygonal Obstacles

The algorithm from **3.3** is extended to account for the introduction of an arbitrary number of polygonal obstacles in the plane. Once the algorithm has completed the approximation of the GM, a call is made to an ancillary function, FindTargetPoints1(), which returns a set of points near the GM but residing in free, unobstructed space. The function first checks if the approximated GM lies within an obstacle, and if it does not, it simply returns the GM. If it does lie inside an obstacle, the function begins an iterative loop where it identifies points in a circle at STEP distance from the GM and checks if they lie inside an obstacle. It increases STEP with each iteration, and ultimately builds a set of points which are not inside obstacles. The number of points to search for is a configurable parameter.

Pseudo-Code for FindTargetPoints1()

```
testpoint ← approximated geometric median
output ← create new, empty ArrayList
MaxDist ← maximum distance within plane
STEP ← MaxDist * 0.01
STEP_SIZE ← STEP
NumTargetPoints ← user configurable parameter, how many points we want to find
NumSearchPoints ← user configurable parameter, how many points on search circle
NumFoundPoints ← 0
num_occlusions ← FindOcclusions()
if num_occlusions > 0:
    while STEP < MaxDist
        build set of points in a circle STEP distance from testpoint
        for each point is search set:
            num_occlusions ← FindOcclusions()
```

```
                        if num_occlusions == 0
                                output ← this point
                                NumFoundPoints++
                        If NumFoundPoints >= NumTargetPoints, break while loop
        end while
        STEP += STEP_SIZE
else:
        output ← testpoint

return output
```

The function which determines if a given point is inside an obstacle is named CheckIfOccluded().

Pseudo-Code for CheckIfOccluded()

```
shape_list ← populate an ArrayList with all the polygonal obstacles
testpoint ← approximated geometric median
for each shape in shape_list:
        if isInShape()
                return true

return false
```

Pseudo-Code for isInShape()

Each shape is a collection of vertices representing the convex hull or a polygon. The vertices are stored in counterclockwise order starting from some arbitrary vertex.

This function traverses this vertex list and checks whether a given input point is counterclockwise to the convex hull of the shape.

```
If it is, then it is inside an obstacle, so return true
Else return false
```

3.5 Heuristic Algorithm Extension – Polygonal Free Space

The algorithm from **3.3** is extended to account for the introduction of an arbitrary number of polygons in the plane which represent free space traversable by people or robot. Once the algorithm has completed the approximation of the GM, a call is made to an ancillary function, FindTargetPoints2(), which returns a set of points representing candidates from each free space polygon. This is accomplished by performing a line search from the center of each shape towards the GM and stopping when we find the last point on the search line inside the shape.

Pseudo-Code for FindTargetPoints2 ()

```
Shapes ← populate the list of shapes
output ← create new hashset
testpoint ← approximated geometric median
For each shape in Shapes:
    if isInShape()
        output ← testpoint
    else:
        perform line search from center of this shape towards testpoint
        add last point that was inside this shape to output

Return output
```

3.6 Alternative Measurement Function Utilizing the Heuristic Algorithm

This algorithm utilizes the framework of the heuristic algorithm, however, it implements a modified version of the **F()** function, named **F2()**. The **F2()** function attempts to minimize the difference between two values. First, the set of target points is divided into two disjoint sets, each set of points being described by one of two descriptors from a discrimination category, for example, male or female. Then, the average Euclidean distance is measured for each of these subsets, against an arbitrary point in the plane. The absolute difference between these two values is then calculated and used as the heuristic for minimization.

3.7 Dynamics Explorer Extensions – Behavior Visualization

Dynamics Explorer is an open source project originally designed as a tool to assist researchers in the field of complex dynamics in visualizing their work. For this thesis, modifications were made to the core functionality of the platform to allow us to visualize the behavior of both Weiszfeld's algorithm as well as the new heuristic algorithm that was developed. The program now generates images in two new classifications:

1. The convergence rate of the GM approximation algorithm for each pixel in the plane.
2. The average distance to all points in ξ for each pixel in the plane.

Additionally, the program will execute either of the GM approximation algorithms and display the results in real-time for any pixel clicked in the image that was generated. It will thus trace the path from the point clicked to the approximated GM, displaying the path as a sequence of connected lines representing each step in the algorithm’s process.

4. Results

4.1 Underlying Parameters of the Experiments

There were two general datasets used during the execution of the experiments:

1. A set of pre-defined target point sets
2. Randomly generated target point sets

The pre-defined sets consisted of 3 and 4 points chosen in such a way as to allow the use of closed-form solutions that produce exact results, along with a set of 5 points chosen as a test bed for the heuristic approximation algorithm to check functionality against a known solution produced by Weiszfeld's algorithm.

The randomly generated sets can produce for analysis target point sets of any size, from 2 points to ∞^{+}, limited only by the host computer's memory and cpu capabilities. These target points are generated using the java.util.Random class, and specifically the Random.nextDouble() function after the parent Random object has been initialized. All the points are thus given random **x** and **y** coordinate values within a user-defined 2d plane. For simplicity's sake, the plane is assumed to have minimum **x** and **y** coordinates of 0.0, and the program user defines maximum values for the **x** and **y** axes. All point computations are performed using the Java **double** primitive, which conforms to the 64-bit IEEE 754 floating point number standard.

Because the points are all randomly generated, as the number of points increases, meta structures or groupings of points become less likely and the points ultimately become evenly dispersed throughout the plane. The data that was gathered over millions of executions of the program shows that given these constraints, as the number of points int the target points set approaches ∞^{+}, the approximated GM converges to the centroid. This is demonstrated by the fact that as the size of ξ grows, the distance between the centroid and approximated GM tends to zero. The relationship between the size of ξ and the distance between the centroid and approximated GM is shown in **Figure 8** and can be expressed as a power function in the form *y = a * x^b,* where a = 0.148217, and b = -0.50193. The predictive power of this model was proven by using it to estimate the *y* value when *x==25000*. The predicted value was *0.091924%*, while the experimentally produced result using 10000 randomly generated test sets was *0.091234%*, a difference of only *0.00069*, or 0.75%.

4.2 Algorithmic Performance – Relative and Absolute Measurements

In terms of relative performance, Weiszfeld's algorithm typically performed better than the heuristic algorithm. The heuristic algorithm performed approximately 2.5 times slower than Weiszfeld's across a test run consisting of the analysis of 10 million randomly generated sets of 5 to 500 target points.

In terms of the absolute number of executions of the **F()** function required to complete the approximation, Weiszfeld's algorithm typically completed with a much lower number of executions, however, edge cases were observed where Weiszfeld's algorithm took a very long time to converge, while the performance of the heuristic algorithm was extremely predictable across all tests.

For a broad sample of performance metrics for both algorithms across a variety of randomly generated scenarios, see **Table 1**. In general, as the size of ξ increases, both algorithms tended to converge more quickly, but Weiszfeld's Algorithm continued to improve faster than the heuristic algorithm. It is also clear that the approximated GM is converging to the centroid, as the distance between them shrinks rapidly.

4.3 Algorithm Convergence Rates – Dependent Factors

The heuristic algorithm converges at a rate primarily dependent on the distance of a given test point to the location of the approximated GM. This rate is influenced by several user-configurable parameters: the number of points to check on the circle during the search phase, the starting step size, the step size multiplier used during successive iterations to shrink the step size, and the ε chosen as the stopping distance. Given known values for the size of the plane and the number of points in the target points set, a savvy user could tune these parameters to produce more optimal results.

A typical example of the visualization of the convergence rate for the heuristic algorithm across an entire sample plane is shown in **Figure 1**. In this example, red represents the fastest convergence rates, while green represents slower convergence rates. **Figure 2** is the same underlying data set of target points, but the image is zoomed out to give a larger perspective on how the convergence rate of the heuristic algorithm is heavily dependent on the distance of any given point to the approximated GM: the further away it is, the slower it is to converge.

Weiszfeld's algorithm, on the other hand, displays complicated behavioral dynamics at a meta level of analysis. Very interesting and complex patterns were observed which show quite clearly that the convergence rate is independent of the distance of a sample point from the approximated GM. Rather, there can be patterns of very rapid convergence resembling "rivers", "whorls", or even "islands" surrounded by a "sea" of much more slowly converging points. Some examples of these exotic behaviors follow.

Figure 3 shows a convergence rate map the Weiszfeld's Algorithm which contained a ring-like structure representing very fast convergence rates, colored in red, embedded in a plane consisting of points which converged much more slowly, colored in yellow and orange.

Figure 4 shows a convergence rate map the Weiszfeld's Algorithm which looks fairly normal on the micro scale, but when zoomed out into **Figure 5**, we can see an anomaly to the southwest of the approximated GM where the convergence rates are exceptionally high, represented by the small red island inside the orange 'teardrop'. In this particular example, that small island converged in literally just a few iterations, while the orange and yellow required dozens of iterations to converge to the approximated solution.

4.4 Visualization of the F() function

As shown in **Figure 6**, Dynamics Explorer can produce images allowing us to visualize the behavior of the F() function when applied to an entire 2d plane. In this example, the F() function is called for each pixel in the plane, and the result is stored and used to color that pixel based on the value that was calculated. The values are mapped to the range $[0, \infty^{+})$, which are represented by the various color bands one can see in the image. Visual inspection of the image appears to show that the F() function exhibits behavior consistent with that of a convex function, as we see a unique global minimum, and the function values appear to change in a continuous manner.

4.5 Visualization of the Polygonal Obstacles Problem

The polygonal obstacles problem describes a situation in which we have a configuration space wherein our service robot must operate, however there exist polygonal obstacles that block portions of that space and render those portions unable to be used.

The heuristic algorithm was extended such that after computing the approximation of the GM, it then performed an analysis of the space to find a new location for the service robot, if necessary. This determination was made based on whether the GM occupied a point that was located inside one of the polygonal obstacles that were defined.

If the GM was not inside an obstacle, then the algorithm terminated, having made the determination that the GM was in unobstructed, free space and was thus available to service customers.

However, if the GM was found to be inside an obstacle, a search was undertaken to find suitable locations nearby as an alternative to the location that was inside an obstacle. This search is performed by radiating outward from the GM as increasing distances until some number of points are found which are not inside any obstacles.

The results of this computation can be seen in **Figure 9**. In this example, the black space is open space that is traversable by people or robots. The grey polygons represent obstacles blocking the open space in some way. The colored dots represent people scattered throughout the space. There are a number of white markers representing critical parts of the computation: the "+" represents the centroid, the hollow "o"s represent the path from the centroid to the original GM, the "X" is the new, alternative GM found to be in the free space, and the lines represent paths to other nearly points in the free space that could also be considered as valid alternates.

4.6 Visualization of the Polygonal Free Space Problem

The polygonal free space problem describes a situation in which we have a configuration space wherein our service robot must operate; however, the free space is broken up into some number of polygonal areas scattered throughout the space. In this scenario, the robot is restricted to occupying only these polygonal free space areas.

The heuristic algorithm was extended such that after computing the approximation of the GM, it then performed an analysis of the space to find locations within each free space polygon to serve as potential locations for the service robot to be placed. To do so, a line search was performed from the center point of each polygon towards the GM, and the program stored the last point that resided inside each polygon.

The results of this computation can be seen in **Figure 10**. In this example, the black space represents space that is not traversable by the robot for some reason. The grey polygons represent free space that is traversable by the robot. The colored dots represent people scattered throughout the space. The white lines represent paths radiating from the GM to alternative locations for the service robot that are in each of the free space polygons. Each polygon has a single point within its mass designated as an alternative location.

4.7 Visualization of the Alternative Measurement Function

As shown in **Figure 7**, Dynamics Explorer can produce images allowing us to visualize the behavior of the F2() function when applied to an entire 2d plane. In this example, the F2()

function is called for each pixel in the plane, and the result is stored and used to color that pixel based on the value that was calculated. The values are mapped to the range $[0, \infty^{+})$, which are represented by the various color bands one can see in the image. Visual inspection of the image appears to show that the F2() function produces results that have no global minimum, rendering it ultimately useless for this research.

5. Conclusion and Future Work

5.1 Heuristic Algorithm

The performance of the heuristic algorithm is highly dependent on a small number of user defined variables. For a given configuration of plane and target points, there may exist optimal sets of parameters. Further research may provide insight into how to best tune those parameters for optimal results.

5.2 Weiszfeld's Algorithm

As shown in the figures included, Weiszfeld's Algorithm displays some complex dynamical behavior. Perhaps that behavior could be analyzed to produce a heuristic or other mechanism by which the algorithm could be seeded with optimal starting locations in order to avoid starting from a location which requires a large number of iterations to converge.

In the figures provided, there is evidence to support the claim that there exist regions of space in a given plane which will converge extremely rapidly for this algorithm. However, at this time, the determining factors which may allow one to predict those regions are unknown. Further research is required to determine if those factors exist, and if they are deterministic in some way.

5.3 Geometric Median

Given the fact that there exists no explicit formula to calculate the GM, rather, some approximation technique must be used, this research calls into question whether it should be abandoned in favor of simply calculating the centroid given the case that the target points are somewhat evenly distributed in the plane and the size of ξ is above some threshold. As shown in **Table 1**, the centroid becomes a very good approximator for the GM rather quickly, as the relative distance between the centroid and GM falls below a proportional difference of 1% once ξ reaches a size of about 250 points. The centroid can be calculated using a single pass through ξ using very basic mathematical operations, whereas approximating the GM requires multiple, generally many, passes through ξ, and requires the use of computationally expensive mathematical operations and advanced techniques. That is, calculating the centroid requires no iterative methods, and an exact formula exists to do so.

5.4 Polygonal Obstacle and Free Space Extensions

For this paper, the algorithmic extensions used to identify alternatives to the GM in the cases where polygonal obstacles are introduced and where polygonal free spaces are introduced were only implemented for the heuristic hill climbing algorithm. However, they could be generalized for use by other algorithms, such as Weiszfeld's. An example scenario would be to use any desired algorithm for approximating the GM, and then calling these extensions as helper functions once the first algorithm completes its calculations.

5.5 Degenerate Cases

While gathering data during these experiments, it was noted that some random scenarios that were generated caused both the heuristic hill climbing algorithm and Weiszfeld's to converge very slowly. The degenerate cases drop off in frequency with increased ξ size. Further investigation may uncover the factors leading to these degenerate cases and provide insight beyond the scope of this paper.

5.6 Applications to Machine Learning

The Geometric Median is used as a subroutine in some machine learning algorithms, including K-Medians clustering. Problems utilizing these algorithms typically work on large datasets, and the results of this research indicate that efficiencies could be found by utilizing the centroid of clusters rather than the Geometric Median, and the resulting additional entropy would be minimal.

5.7 Ethical Considerations

Ultimately, the point of this research is to establish a quantitative methodology for placement of service robots that can be shown to be non-discriminatory using some form of rigorous analysis, that is, to be ‘fair’ in some way. Certainly, the method described in this paper, that of using the Geometric Median as an idealized location for a service robot, is not the only method one could use that is measurably ‘fair’. Other methods could be evaluated, and their results gauged against these.

One may also consider the idea that a measurably ‘fair’ methodology may not be preferred by actual customers of such a robot. Social experiments to evaluate people’s perception could be performed to provide insight into how people perceive fairness and discrimination in their interactions with service robots.

The construction of algorithms specifically to address issues such as the ethics of discrimination introduce the possibility that these algorithms may be used for purposes contradictory to their stated intent. Indeed, it would be trivial to use such an algorithm to purposely place a service robot in such a way as to discriminate in favor of a particular person or group. Further research could be performed to provide tools for individuals to analyze the behavior of service robots they interact with in order to determine if they are programmed to act in ways which may be discriminatory or violate other ethical expectations.

Figures

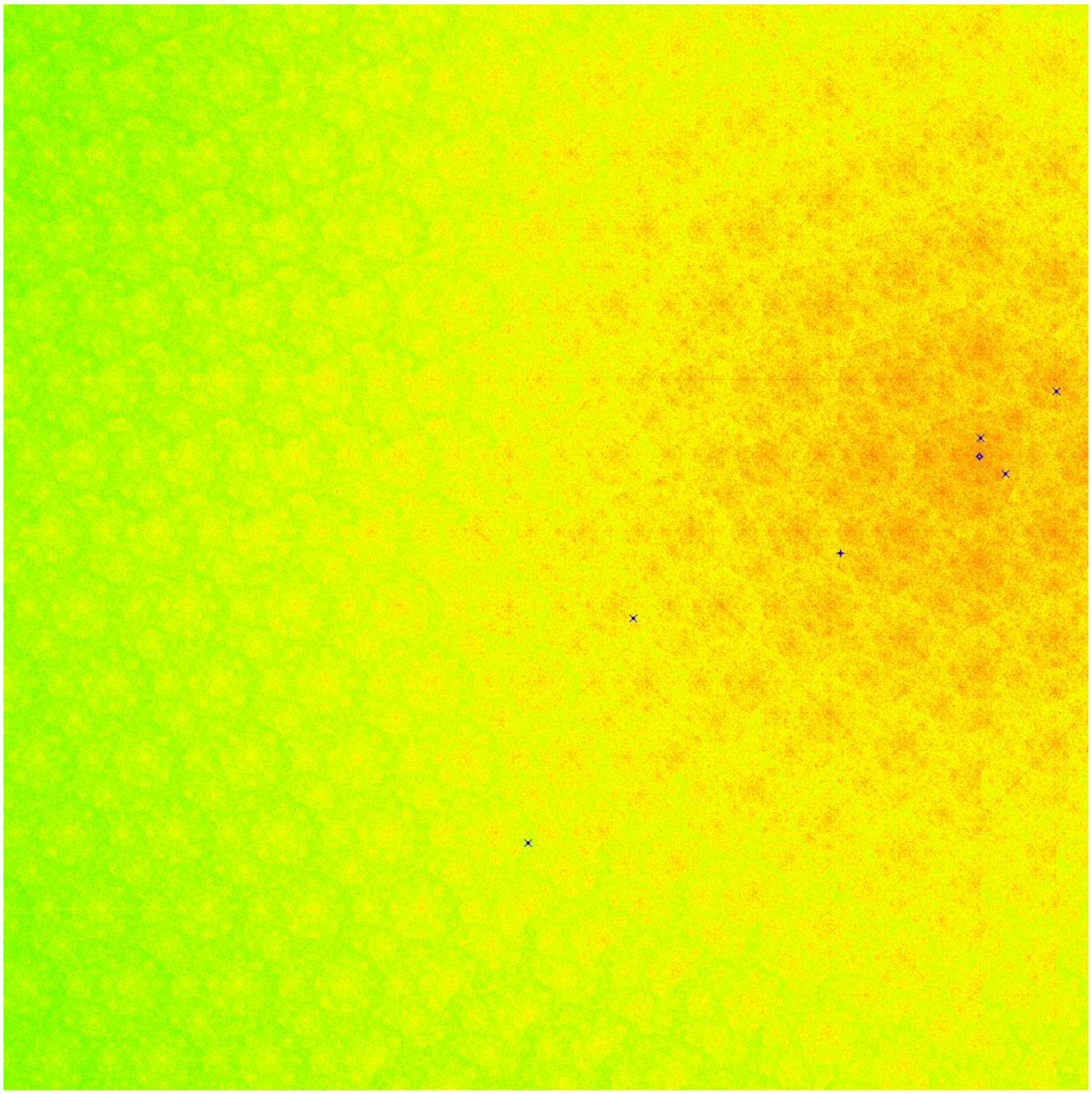

Figure 1: Convergence rate map for Heuristic Algorithm, 5 target points.

Red coloration shows faster convergence, while green coloration is slower.

The octagonal pattern is bias introduced by the algorithm being configured to examine 8 points in a circle whenever the search subroutine is called.

x = target points   + = centroid   ♦ = GM

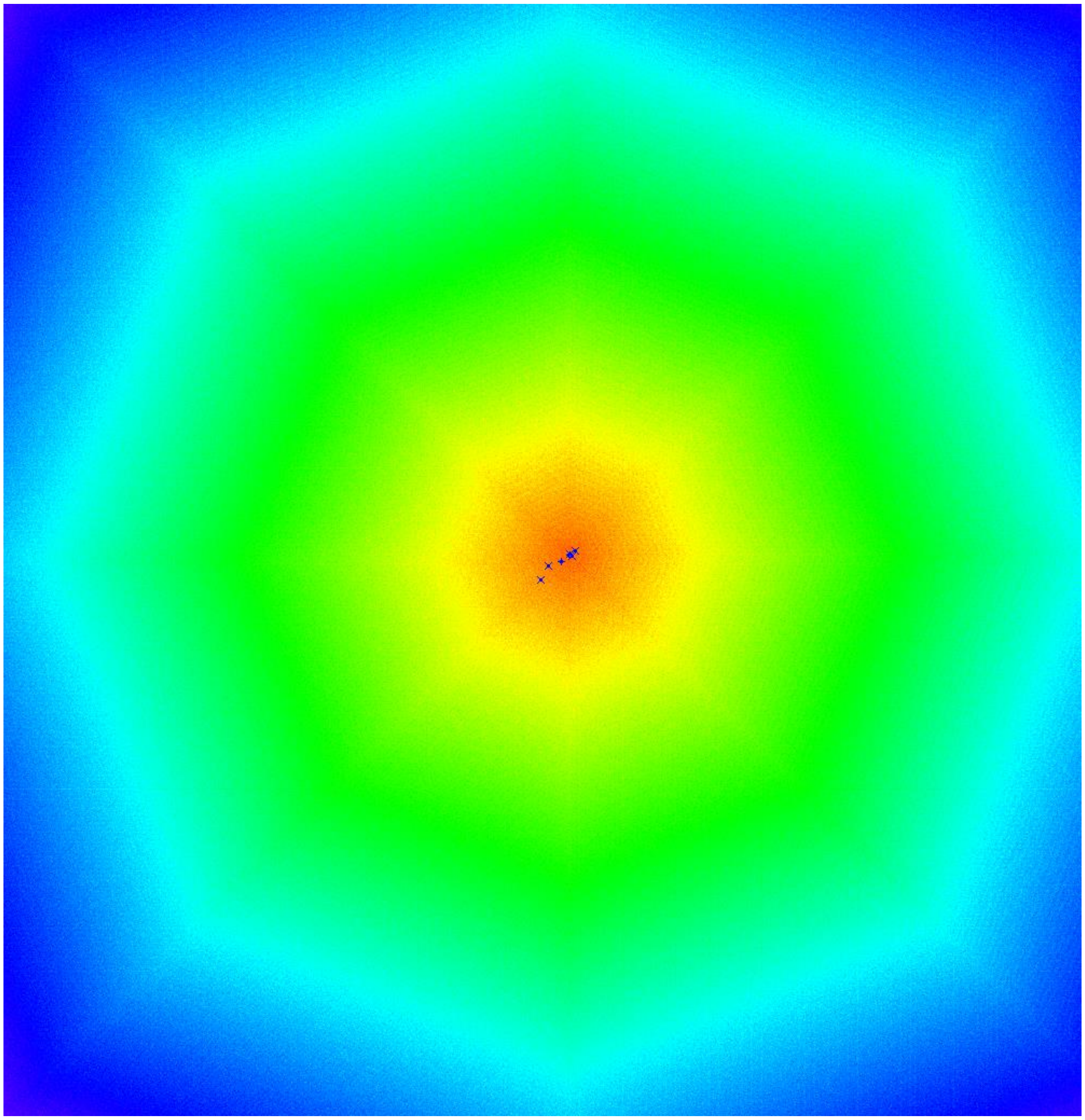

Figure 2: Convergence rate map for Heuristic Algorithm, 5 target points, zoomed out to show increase in convergence rate due to distance from GM.

Red coloration shows faster convergence, while green coloration is slower.

x = target points + = centroid ♦ = GM

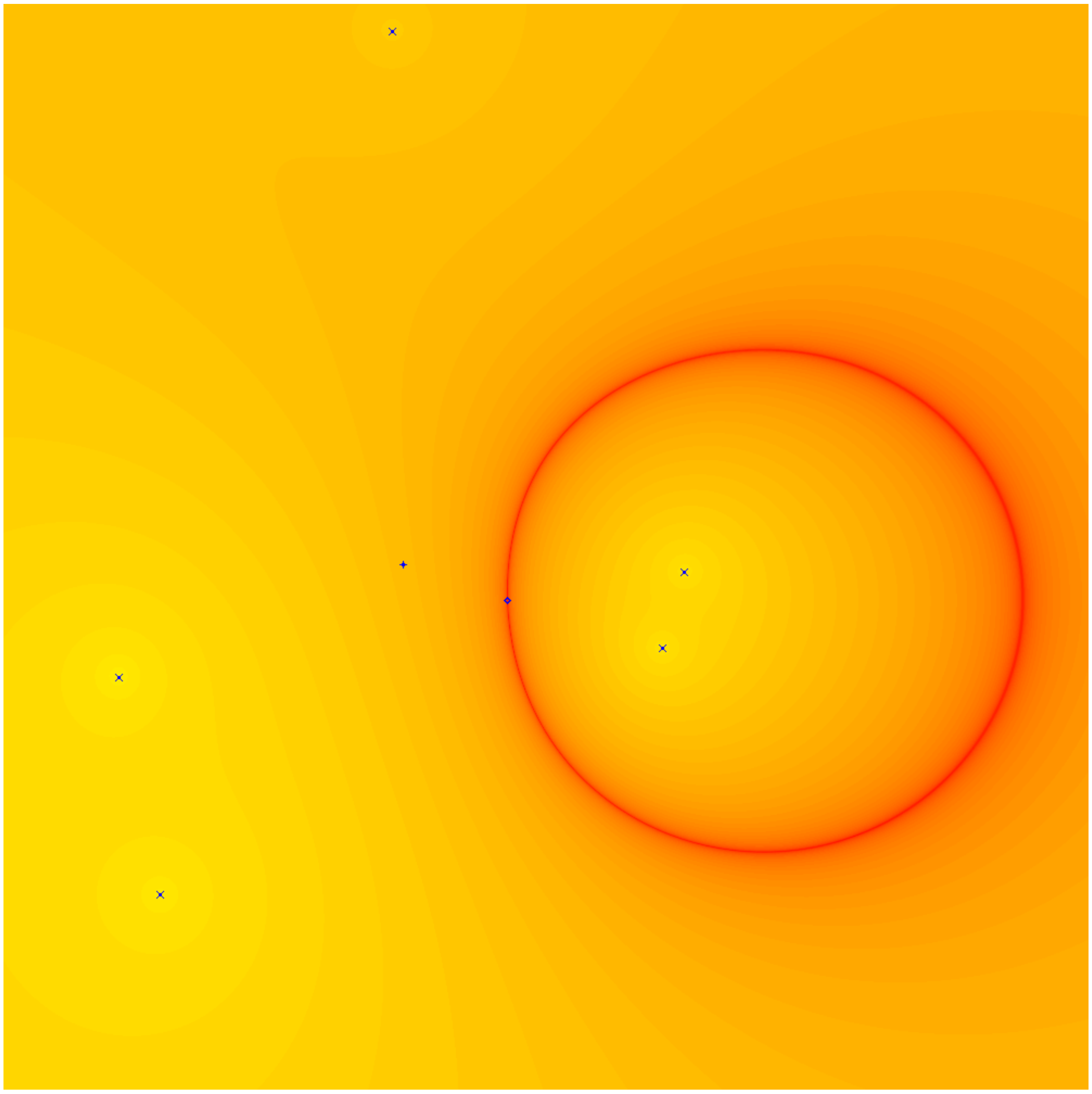

Figure 3: Convergence rate map for Weiszfeld’s Algorithm, 5 target points.

The dynamics show a ring-like structure of very fast convergence, shown in red, while most of the plane converges much more slowly, as indicated by yellow and orange hues.

x = target points + = centroid ♦ = GM

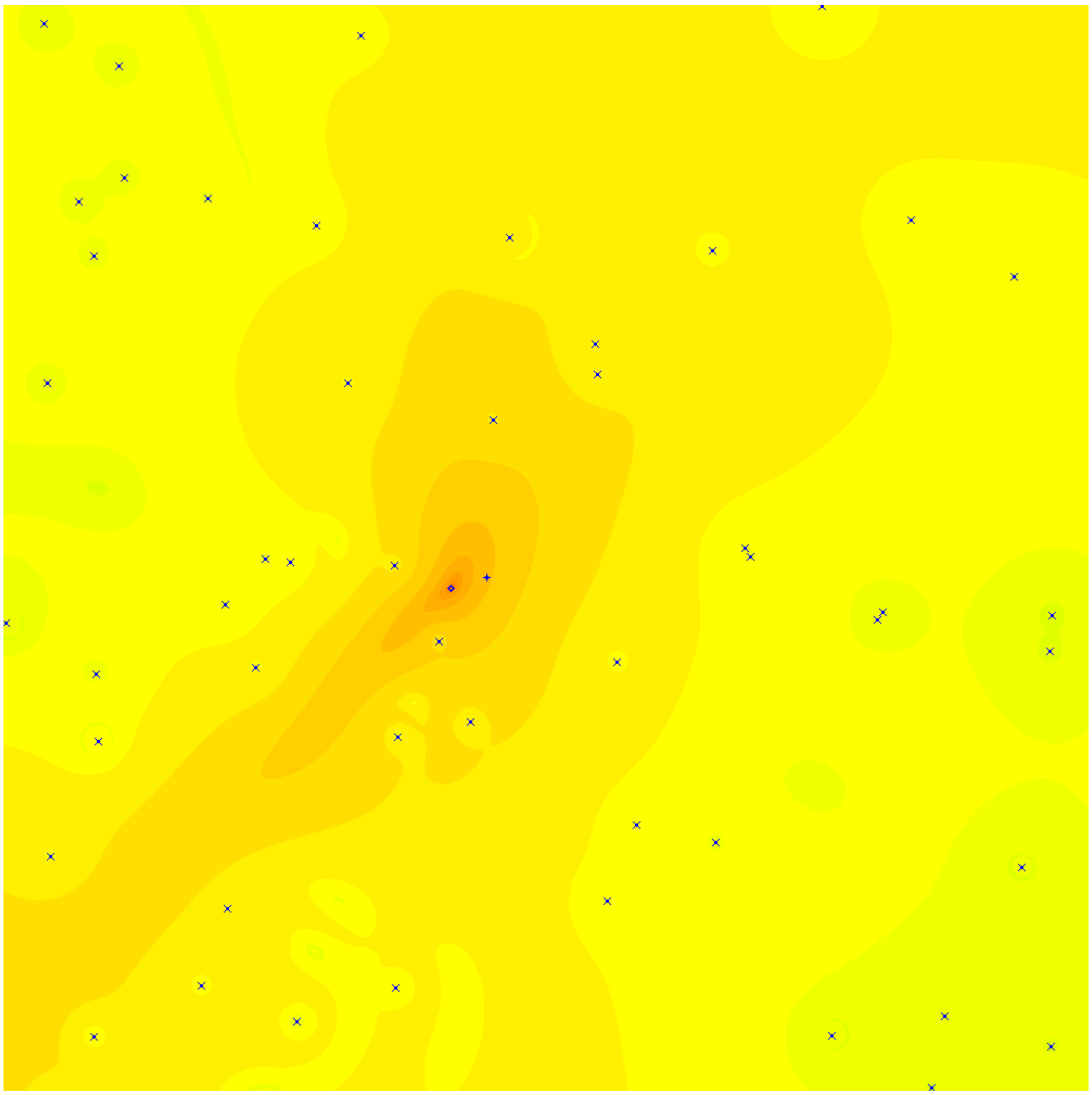

Figure 4: Convergence rate map for Weiszfeld's Algorithm, 50 target points.

x = target points + = centroid ♦ = GM

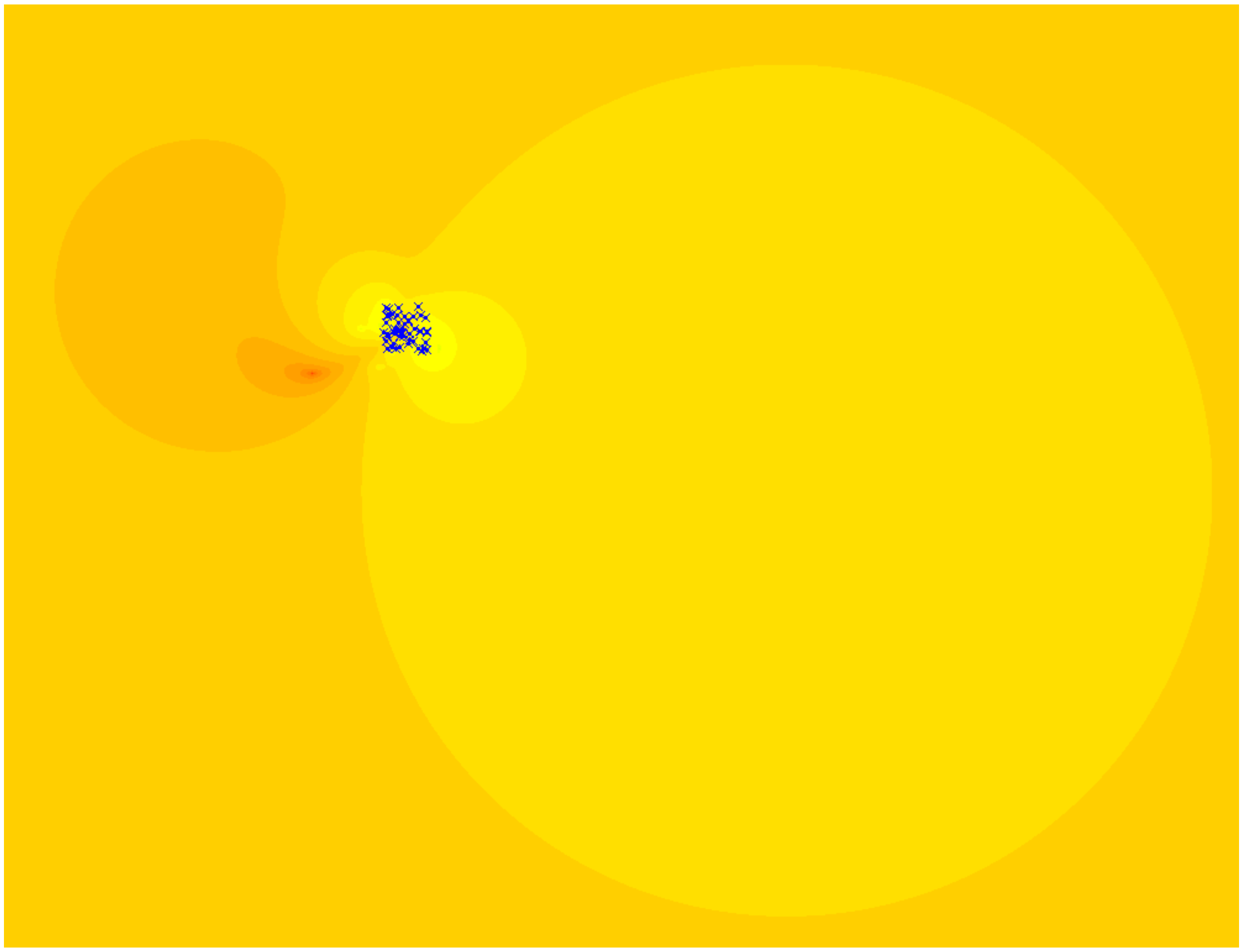

Figure 5: Convergence rate map for Weiszfeld's Algorithm, 50 target points, zoomed out.

At this level of zoom, we can see an anomaly to the southwest of the approximated GM where convergence rates are exceptionally high, represented by the small island of red inside an orange 'teardrop'.

x = target points   + = centroid   ♦ = GM

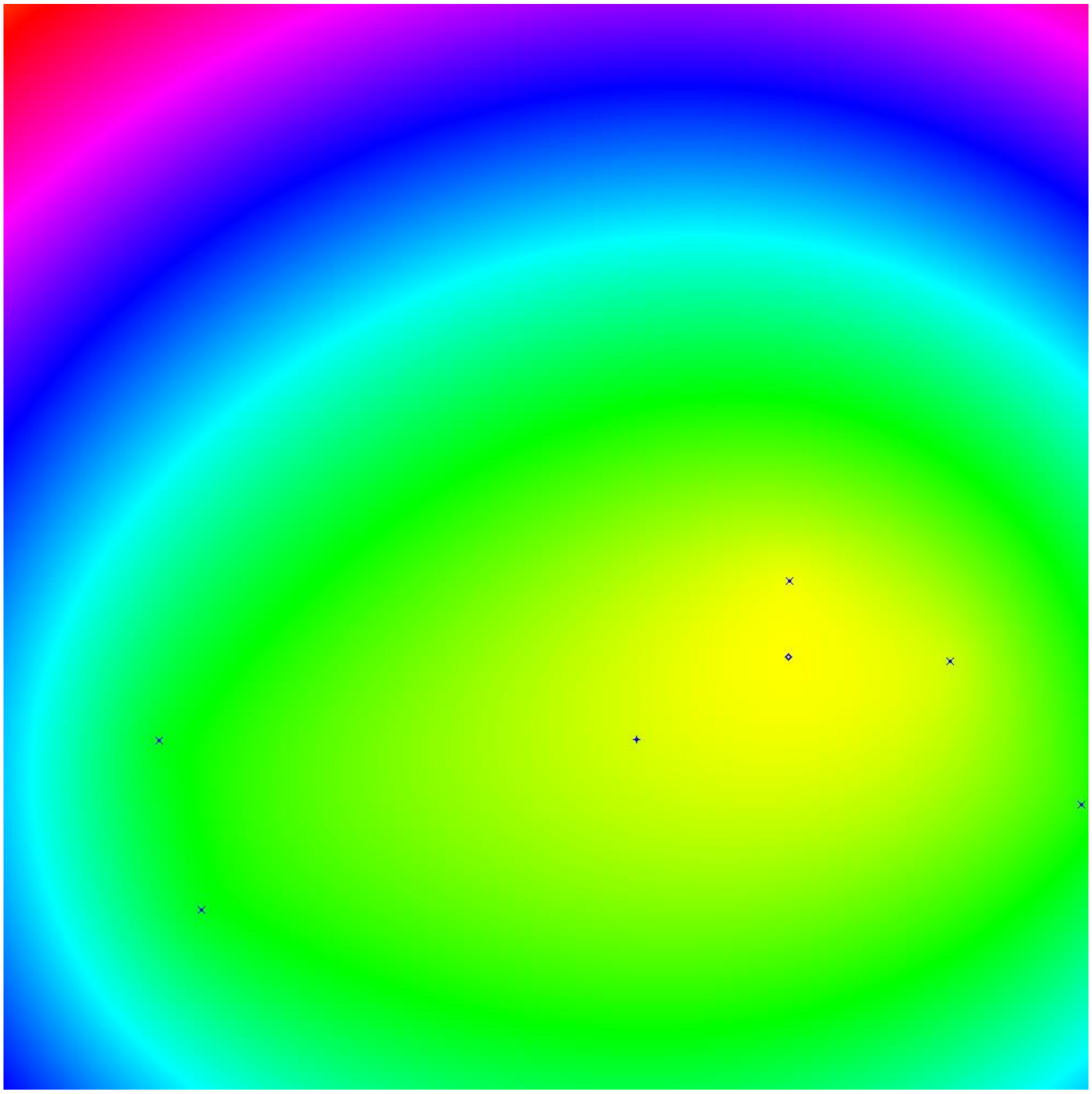

Figure 6: Each pixel is colored based on the value of the F() function for that particular point.

Yellow coloration shows low F() values, increasing in a linear fashion out to the red band representing the highest F() values.

x = target points + = centroid ♦ = GM

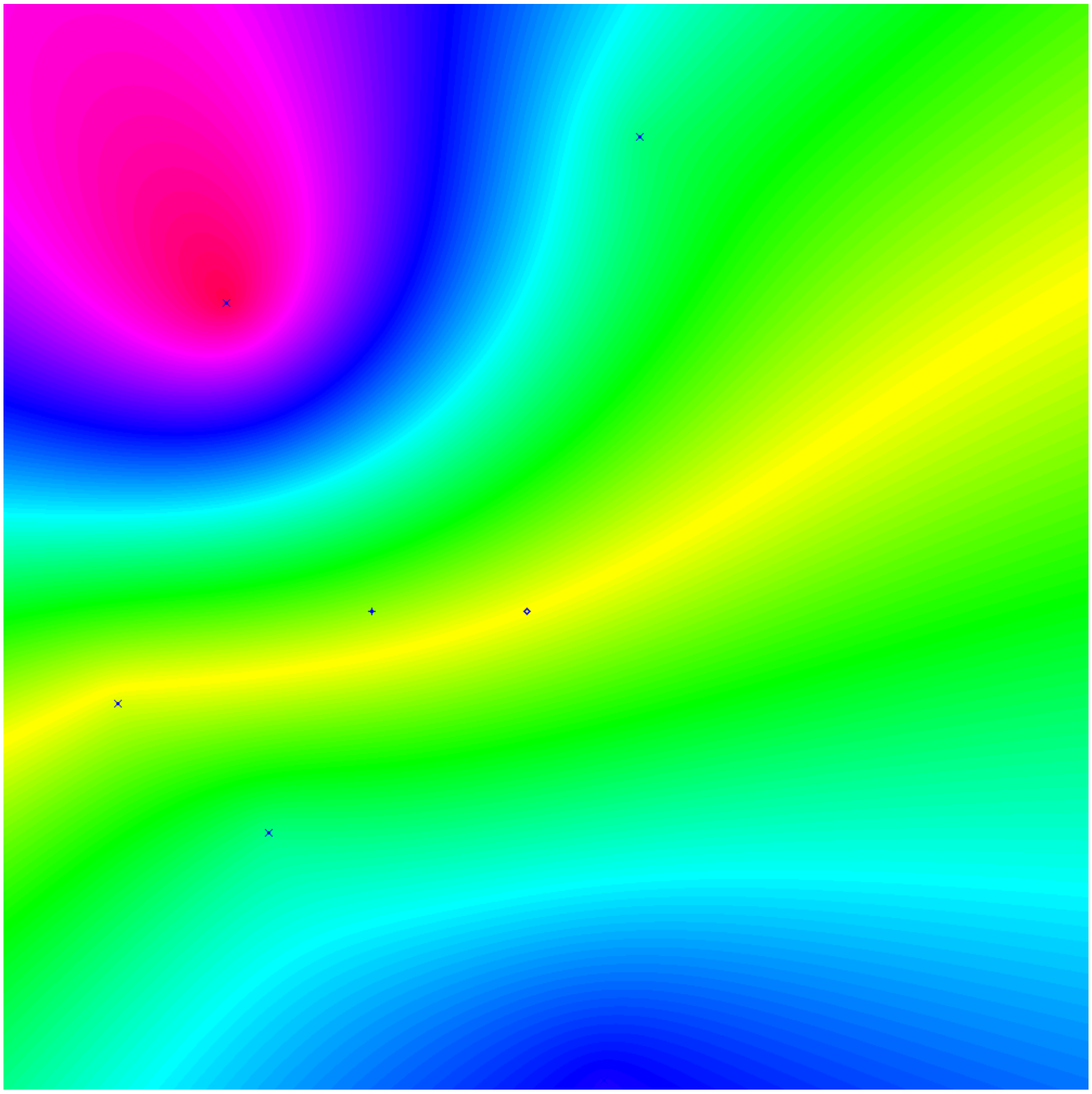

Figure 6: Each pixel is colored based on the value of the F2() function for that particular point.

Yellow coloration shows low F2() values, increasing in a linear fashion out to the red band representing the highest F2() values.

Examination of points within the yellow band shows multiple identical values, demonstrating that this function did not have a unique global minimum within the visible portion of the plane.

x = target points   + = centroid   ♦ = GM

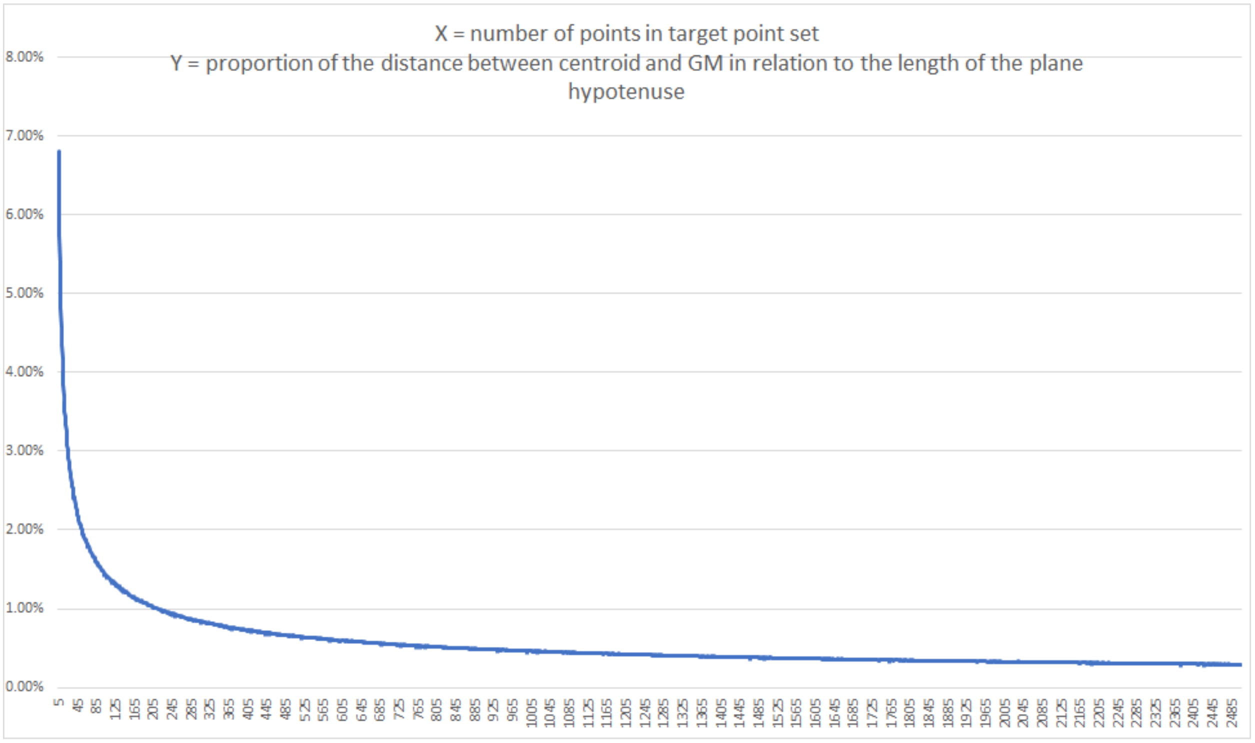

Figure 8: The relationship between the size of the target point set on the x-axis, and the distance between the calculated centroid and the approximated GM, represented on the y-axis by the ratio of the distance between them as a proportion of that distance to the hypotenuse of the plane the tests were generated within.

For a given plane, as the size of the target point set approaches ∞^+, the approximated GM converges to the location of the centroid.

The shape of the curve is that of a power function, such that the initial convergence is very fast, and then the rate slowly tapers off.

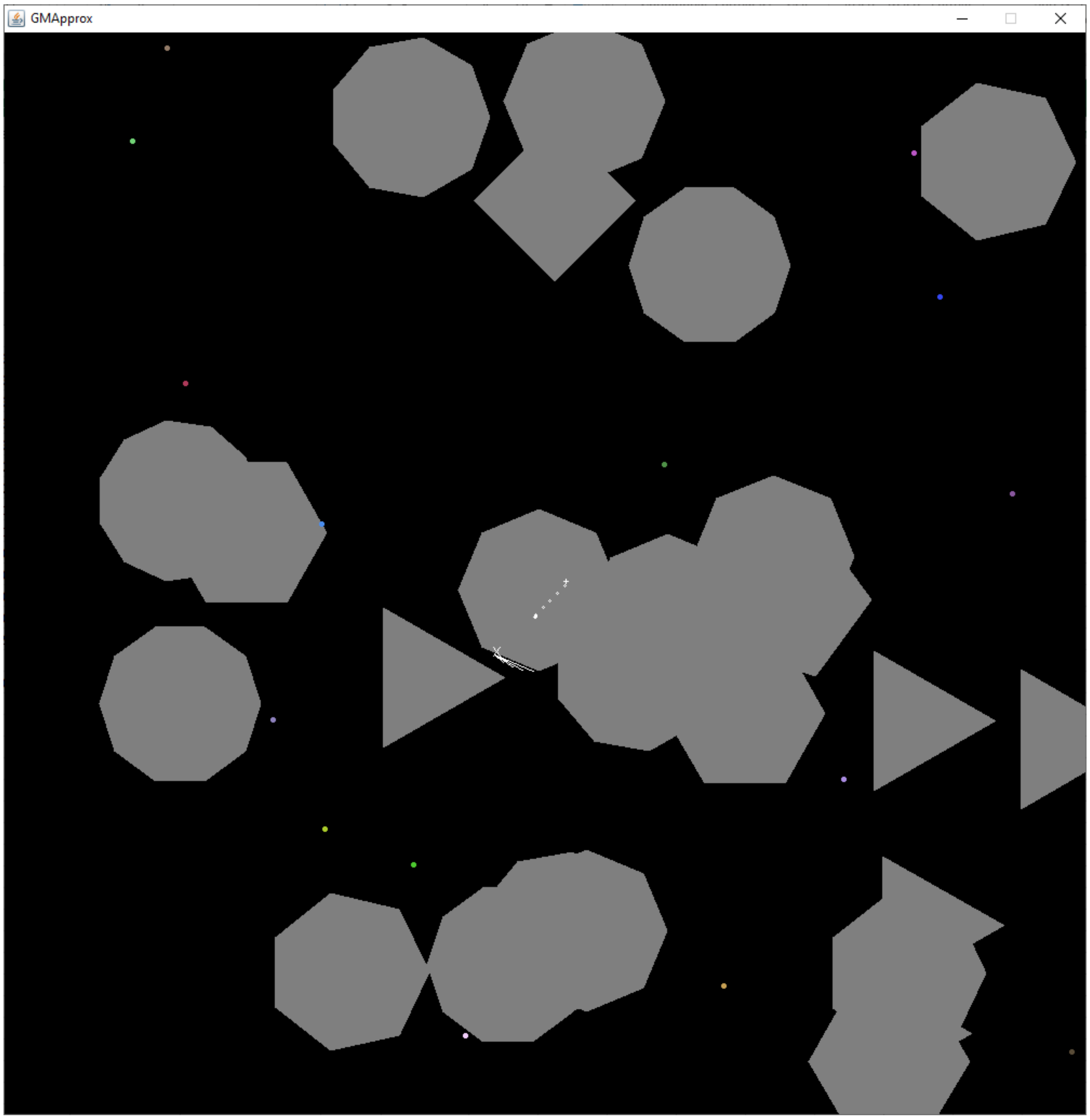

Figure 9: Visualization of the solution to the "polygonal obstacles" problem.

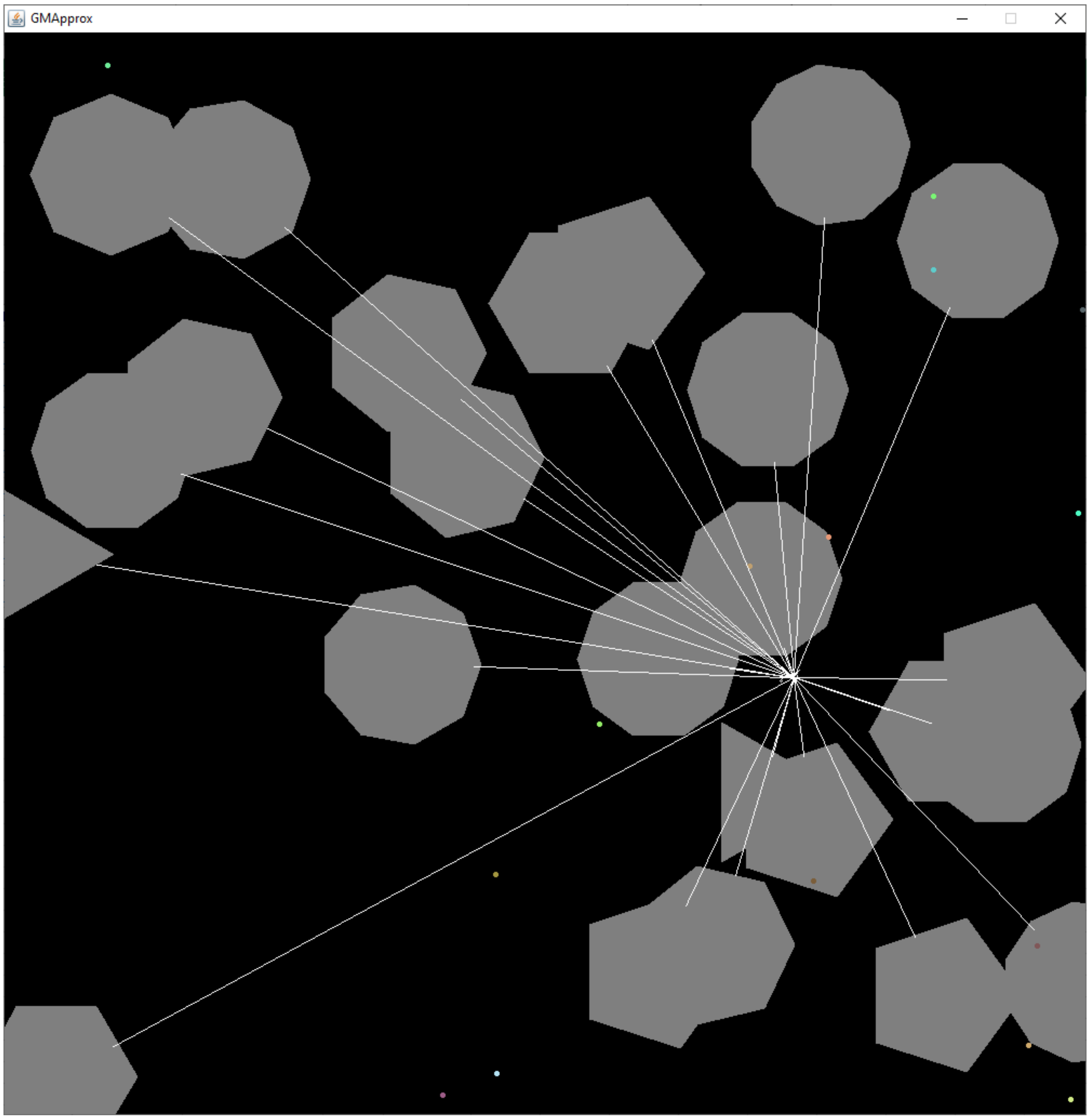

Figure 10: Visualization of the “polygonal free space” problem.

Tables

# Points	H Checked	W Checked	H Dist	W Dist	H F() Diff	W F() Diff
5	138.072931	57.605846	193.071743	192.750065	0.032116	0.032517
10	129.175582	45.895784	135.252494	136.225233	0.013958	0.0141
25	120.837968	34.631083	84.740626	84.111517	0.005389	0.005211
50	115.472334	29.162086	59.068263	57.078377	0.002574	0.002406
100	111.280942	26.367312	41.447509	41.246903	0.001266	0.001256
250	107.227232	23.692076	26.088124	26.359305	0.000499	0.000508
500	104.063903	22.318934	18.621822	18.570915	0.000254	0.000252
750	102.106024	21.613461	15.149446	15.021718	0.000167	0.000165
1000	100.728498	21.262479	13.018245	13.045711	0.000124	0.000125
2500	96.271137	20.208025	8.325911	8.241502	0.000051	0.00005

Table 1: Subset of data captured during computation of approximation algorithms. 10,000 randomly generated scenarios for each ξ.

Fields:

# Points: the number of points in given ξ

H Checked: average number of points checked using the F() function for the heuristic algorithm to complete its approximation of the GM

W Checked: average number of points checked using the F() function for Weiszfeld's Algorithm to complete its approximation of the GM

H dist: average absolute distance between the centroid and GM using heuristic algorithm

W dist: average absolute distance between the centroid and GM using Weiszfeld's algorithm

H F() diff: average ratio of the difference between the F() value for the calculated centroid, and the F() value for the GM using heuristic algorithm

W F() diff: average ratio of the difference between the F() value for the calculated centroid, and the F() value for the GM using Weiszfeld's algorithm

References

[1] J Wirtz, P Patterson, W. Kunz, T. Gruber, V. Lu, S. Paluch, and A. Martins, “Brave new world: service robots in the frontline,” *Journal of Service Management*, Vol. 29, No. 5, pp 907-931, 2018.

[2] H.W. Kuhn, “A note on Fermat's problem,” Mathematical Programming *4*, pp 98–107, 1973.

[3] E. Weiszfeld, “Sur le point par lequel la somme des distances de*n* points donnés est minimum,” *Tohoku Mathematics Journal 43*, pp 355–386, 1937.

[4] S. Park, W. Yu, and J. Cho, “A user reactivity research for improving performance of service robot,”. *URAI 2011 - 2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence*, 2011.

[5] Y. Kobayashi, M. Gyoda, T. Tabata, Y. Kuno, K. Yamazaki, M. Shibuya, and Y. Seki, “Assisted-Care Robot Dealing with Multiple Requests in Multi-party Settings”, *6th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2011.

[6] R. Stock and M. Merkle, “A service Robot Acceptance Model: User acceptance of humanoid robots during service encounters,” *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp 339-344, 2017.

[7] S. Samarakoon, H. Sirithunge, M. Muthugala, and A. Jayasekara, ”Proxemics and Approach Evaluation by Service Robot Based on User Behavior in Domestic Environment,” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[8] E. Belas and C. Yu, “A Note on the Weiszfeld-Kuhn Algorithm for the General Fermat Problem,” *Managemenr Science Research Report No. MSRR 484, Carnegie-Mellon University*, 1982.

[9] K. Aftab, R. Hartley, J. Trumpf, “Generalized Weiszfeld Algorithms for Lq Optimization,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 37, Issue 4, 2015.

[10] C. Kasemtaweechok and W. Suwannik, “Training Set Reduction using Geometric Median”, *15th International Symposium on Communications and Information Technologies (ISCIT)*, 2015.

[11] A. Kogler, P. Traxler, “Efficient and Robust Median-of-Means Algorithms for Location and Regression”, *18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2016.

[12] X. Hua, Y. Cheng, H. Wang, Y. Qin, Y. Li, “Geometric means and medians with applications to target detection”, *IET Signal Processing*, Vol. 11, Issue 6, 2017.

[13] C. Kasemtaweechok and W. Suwannik. “Prototype Selection for k-Nearest Neighbors Classification Using Geometric Median,” *ICNCC '16*, 2016.

[14] M. Cohen, Y. Lee, G. Miller, J. Pachocki, A. Sidford, “Geometric Median in Nearly Linear Time”, *STOC ’16 proceedings of the forty-eight annual ACM symposium on Theory of Computing*, pp 9-21, 2016.

[15] J. Radon. “Mengen konvexer Körper, die einen gemeinsamen Punkt enthalten,” *Mathematische Annalen*, Vol. 83, pp 113-115, 1921.

[16] C. Bajaj. "Proving geometric algorithms nonsolvability: An application of factoring polynomials". *Journal of Symbolic Computation*. Vol. 2, pp 99–102, 1986.

[17] M. Protter, C. Morrey Jr, *College Calculus with Analytic Geometry* (2nd ed.), 1970.

Appendix A

Pseudo-Code for Weiszfeld's Algorithm

Complexity: Time is super-linear in the number of elements in ξ. Space is linear in the number of elements in ξ.

Input: a seed point to serve as the first iterate

Output: a data structure containing the approximated GM point, the count of how many iterates were produced, and a LinkedList containing all the iterates

```
$\varepsilon \leftarrow$ <some configured value, very small and positive>
$\xi \leftarrow$ target point set
diff $\leftarrow \infty^+$
P $\leftarrow$ starting point      <usually the centroid, but if null, generate a random starting point>
sum_w $\leftarrow$ 0
$Q_k \leftarrow$ new point, (0,0)
While diff > $\varepsilon$
    for each point in $\xi$:
        Q $\leftarrow$ 'this' point in $\xi$
        if P == Q, execute subroutine to break out of errant minimum (**Appendix J**)
        else, continue
        d $\leftarrow$ dist(Q, P)              <get Euclidean distance>
        w $\leftarrow$ 1/d                     <calculate this weight>
        sum_w $\leftarrow$ sum_w + w           <accumulate sum_q>
        $Q_k \leftarrow Q_k$ + (Q * w)         <vector addition, multiplication of Q's values>
    $Q_k \leftarrow Q_k$ / sum_w
    diff $\leftarrow$ dist($Q_k$, P)
    P $\leftarrow Q_k$
end While
```

Appendix B

Pseudo-Code for Heuristic Algorithm

Complexity: Time is linear in the number of elements in ξ and dominated by a constant factor which is influenced by the relationship between multiple parameters, including the initial STEP_SIZE, the STEP_DELTA, and the number of points on the search circle. Space is linear in the number of elements in ξ.

Input: a seed point to start from, generally the centroid

Output: a data structure containing the approximated GM point, the count of how many iterations were processed, the count of how many points were examined using the F() function, the count of how many iterates were produced, and a LinkedList containing all the iterates

```
Centroid ← calculate centroid using known formula
ClosestPoint ← Centroid                              <initial starting point>
ClosestPointF ← F(ClosestPoint)
CheckedPoints ← add ClosestPoint to hashset
STEP_DELTA ← CONFIGURED BY USER                      <value between 0 and 1>
STEP_SIZE ← CONFIGURED BY USER
OuterLoop:
While STEP_SIZE > $\varepsilon$
    iteration++
    NewClosestPoint ← TRUE
    While NewClosestPoint == TRUE
        NewClosestPoint ← FALSE
        Neighbor ← GetNextPoint()
        NeighborF ← F(Neighbor)
        If NeighborF < ClosestPointF
            NewClosestPoint ← TRUE
            diff = dist(ClosestPoint, Neighbor)
            ClosestPoint ← Neighbor
```

```
                ClosestPointF ← NeighborF
                CheckedPoints ← add ClosestPoint to hashset
                movecount++
                if diff < $\varepsilon$
                    break OuterLoop
    end While
    STEPSIZE ← STEP_SIZE * STEP_DELTA
end While
```

Appendix C

Pseudo-Code for GetNextPoint()

Complexity: Time is linear in the number of elements in ξ. Space is linear in the number of elements in ξ.

Input: the current point, a HashSet of points that have already been checked, the STEP_SIZE, and a path hint

Output: a data structure containing the next point in the iterate sequence, the F() value for that point, the number of points checked during the search process, and angle from the current point to the next point representing the path hint

```
If no existing path hint (move failed during last iteration):
    Search in a circle around current point. Using polar coordinated, divide the circle into some pre-configured number of arcs and check points at the end of each arc. This gives us some number of samples in a circular pattern around our current location in a plane.
    Check each of these sample points using the F() function to calculate the sum Euclidean distance to all points in the target point set.
    If we find a point with a lower F() value than our current point:
        Use binary decomposition to search near this new candidate for a better match.
        Return best match found.
    If not find a better point:
        Return null to calling function so iteration can fail and STEP_SIZE can be reduced for next iteration

If existing path hint (last move was successful):
    Move in the same direction as last iteration to get a new candidate, and examine clockwise and counterclockwise neighbors of this candidate
    Check F() values of these potential candidates, and return best match, if found, or null if not found
```

Appendix D

Pseudo-Code for F()

Complexity: Time is linear in the number of elements in ξ. Space is linear in the number of elements in ξ.

Input: the point to test and ξ (the set of all target points) to test against

Output: double value representing sum of the distance from the test point to each point in the target point set

```
Given a candidate point and ξ:
    output ← 0
    For each point in ξ:
        output ← output + dist(candidate, this point)
    return output
```

Appendix E

Pseudo-Code for dist()

Complexity: Time is linear in the number of dimensional components of the points. Space is linear in the number of dimensional components of the points.

Input: two points

Output: double value representing the Euclidean distance between the two points

Given two points, p and q, return the Euclidean distance between them, using the Pythagorean Theorem:

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Appendix F

Pseudo-Code for FindTargetPoints1()

Complexity: Time is linear in NumSearchPoints. Space is linear in NumTargetPoints.

Input: seed point to start searching from, and the number of nearby points to find

Output: ArrayList of suitable points

```
testpoint ← approximated geometric median
output ← create new, empty ArrayList
MaxDist ← maximum distance within plane
STEP ← MaxDist * 0.01
STEP_SIZE ← STEP
NumTargetPoints ← user configurable parameter, how many points we want to find
NumSearchPoints ← user configurable parameter, how many points on search circle
NumFoundPoints ← 0
num_occlusions ← FindOcclusions()
if num_occlusions > 0:
    while STEP < MaxDist
        build set of points in a circle STEP distance from testpoint
        for each point is search set:
            num_occlusions ← FindOcclusions()
            if num_occlusions == 0
                output ← this point
                NumFoundPoints++
            If NumFoundPoints >= NumTargetPoints, break while loop
    end while
    STEP += STEP_SIZE
else:
        output ← testpoint

return output
```

Appendix G

Pseudo-Code for CheckIfOccluded()

Complexity: Time is linear in the number of elements in the shape list. Space is linear in the number of elements in the shape list.

Input: set of polygonal obstacles, and a point to test

Output: Boolean value representing whether the test point is inside an obstacle or not

```
shape_list ← populate an ArrayList with all the polygonal obstacles
testpoint ← approximated geometric median
for each shape in shape_list:
    if isInShape()
        return true

return false
```

Appendix H

Pseudo-Code for isInShape()

Complexity: Time is linear in the number of vertices in the shape. Space is linear in the number of vertices in the shape.

Input: a polygonal shape as a set of vertices, and a point to test

Output: Boolean value representing whether the test point is inside the given shape

Each shape is a collection of vertices representing the convex hull of a polygon. The vertices are stored in counterclockwise order starting from some arbitrary vertex.

This function traverses this vertex list and checks whether a given input point is counterclockwise to the convex hull of the shape.

If it is, then it is inside an obstacle, so return true

Else return false

Appendix I

Pseudo-Code for FindTargetPoints2 ()

Complexity: Time is linear in the number of elements in the shape list. Space is linear in the number of elements in the shape list.

Input: point representing the approximated GM

Output: HashSet of suitable points

```
Shapes ← populate the list of shapes
output ← create new hashset
testpoint ← approximated geometric median
For each shape in Shapes:
    if isInShape()
        output ← testpoint
    else:
        perform line search from center of this shape towards testpoint
        add last point that was inside this shape to output

Return output
```

Appendix J

Pseudo-Code for Weiszfeld's Algorithm: Belas and Yu Modification

Complexity: Time is linear in the number of elements in ξ. Space is linear in the number of elements in ξ.

Input: a_k, member of ξ

Output: Point representing next iterate

```
$R_k = \Sigma$ {over i, from 1 to m, $i \neq k$} $((w_{i)} / (dist(\xi_i , a_k)) * (\xi i - ak))$
$Norm_k = norm_2(R_k)$
$w_k = 1/ \varepsilon$

if $(w_k \geq Norm_k)$
    return $a_k$
else
    <use bisection method to find next iterate>
    lambda = 1.0
    descent = $a_k$ + lambda * $R_k$
    while lambda $\geq \varepsilon$ and F(descent) $\geq F(a_k)$
        lambda = lambda / 2
        descent = $a_k$ + lambda * $R_k$
    end while
    if F(descent) < $F(a_k)$
        return descent
```