8-2016

A Computational Geometric and Graph Theoretic Approach to Reducing Dimensionality on Raster Data Problems

Matthew James Robert Bachstein
University of Tennessee, Knoxville, mbachste@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Matthew James Robert Bachstein entitled "A Computational Geometric and Graph Theoretic Approach to Reducing Dimensionality on Raster Data Problems." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.

Charles R. Collins, Major Professor

We have read this thesis and recommend its acceptance:

Michael Berry, Abner Salgado

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

A Computational Geometric and Graph Theoretic Approach to Reducing Dimensionality on Raster Data Problems

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Matthew James Robert Bachstein

August 2016

I dedicate this work to my family, who pushed me toward higher education. For my parents and sister who acted as a sounding board for new ideas and concepts. To my wife, whose love and support kept me going. And also to my grandparents Charles and Linda Upton, and Winfred and Patricia Bachstein.

Acknowledgements

I would like to thank all those whose love and support made this work possible. I also would like to acknowledge my committee, Dr. Michael Berry, and Dr. Abner Salgado for their guidance and insight, and my chair Dr. Charles Collins whose guidance and mentorship have been instrumental in my growth as an aspiring academician.

Abstract

Large scale mathematical models often involve a trade off between computational length and detail. In general, the more detailed the data, the more time it takes for the model to process. Models that use geographic scale data are particularly susceptible to this inflation; fine resolution data (on the order of m^2 [meters squared]) brings great benefits, but demolishes the computation time. This thesis presents a method for reducing the dimensionality of large scale data in a systematic manner to maximize the benefits of fine resolution data while minimizing the computational time increase, then applying the method to a simulated invasive species problem using geographic data.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Using detailed geographical data to model local movement over a wide area ofttimes involves a number of individual data points that would make most standard computational models infeasible with the number of calculations necessary. One of the most common methods of dealing with this problem is increasing the 'resolution' of the data by aggregating adjacent points together and averaging their values, accepting inhomogeneity for each larger data point for an increase in computational speed.

As an example, the approach taken by Carr, Collins, Corn, et. al [2] modeling the spread of feral hogs started with amalgamating the data from $30m^2$ squares, the natural resolution, into $240m^2$ squares (8x8). This had the effect of turning just shy of two hundred million ($199.3 * 10^6$) data points into just over three million points ($3.125 * 10^6$). The data was then further condensed into $1920m^2$ squares, reducing the total number of data points to just under forty-nine thousand ($48,930$). As stated previously, each final data point is an average of an average of the original values, losing a lot of the fine detail in the process. Visually, we can see the difference in Figure 1.1 as compared to Figure 1.2, most noticeable in the upper right where many of the points with values around 90 have been washed out.

This was not much of a concern since the authors were creating a logistic probability model, but for a more deterministic model such a reduction in the data resolution and fidelity might not be wanted; inhomogeneity combined with losing data adds an element of uncertainty that would not be present otherwise.

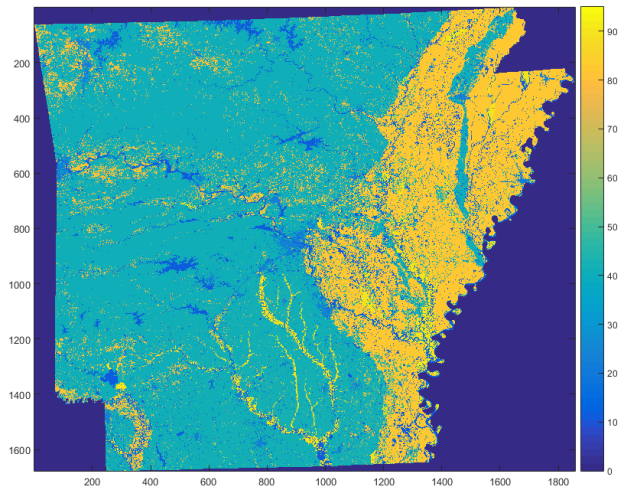

Figure 1.1: NLCD data after 240x240 averaging

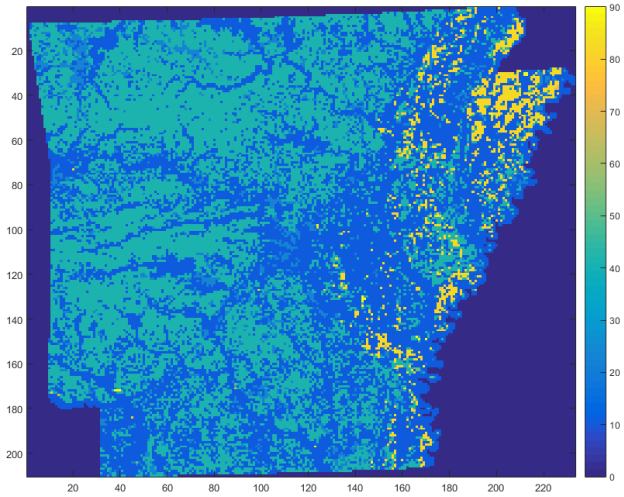

Figure 1.2: NLCD data after 1920x1920 averaging

The question becomes twofold: Can a method be devised with no averaging that does not increase the computation time by an unacceptable amount? And, what mathematical conditions should this method satisfy? To try and address these questions, this work develops a method that aggregates only like valued data into clusters, parameterized on the wanted convexity tolerance, then links the clusters as nodes on a weighted graph before running a simulation, based on a diffusion model. This allows for needing only the assumption that the rate of spread through out each node is a constant rate, rather than having to account for the variance due to non-homogeneous and non-convex clusters. Aggregating the points into clusters is a straightforward process using breadth-first search to traverse across the entire data array. Then, the problem breaks down into two parts, first measuring the convexity for each, and node then attempting to maximize its convexity if it falls below a threshold, by finding a 'critical axis' along which the separating node will give two nodes each with convexity above the threshold. And second, creating a graph from the resulting clusters whose parameters are chosen in a manner that provides the closest match to known data. Additionally, turning the question from an areal diffusion problem into a graph theory problem allows for application of some theorems and algorithms in that field, for instance finding shortest paths, minimum spanning trees, and maximal cuts. These would be useful in secondary applications such as optimal resource allocation for invasive species control.

In Chapter 2, the mathematical theories that form the foundation of the method are addressed, first looking at how to determine an adequate metric for measuring the convexity of a set and second determining criteria to create the graph for the simulation. Chapter 3 discusses implementation details about the program and displays several examples of its execution on some sample datasets: predetermined shapes, random data, and actual geographic data. Chapter 4 discusses some of the advantages and drawbacks of the method as well as some possibilities for future work.

Chapter 2

Theory and Methods

Investigation into this problem yielded two main areas of research; computational geometry, and graph theory. Computational geometry is a subfield of computer science studying algorithms that can be expressed in terms of mathematical geometry. Since one of the core problems is to determine a useful computed metric to categorize a clusters shape, this seemed a natural field to inquire within. Most of the research along the lines of 'shape analysis' came out of the field of landscape design & architecture, with those concepts then being brought over and further developed in geography research. For this problem, shape is important because we want each cluster to be convex 'enough;' how much is left to the user. This is to satisfy the convexity condition for an ordinary differential equation solution.

2.1 Computational Geometry

The first problem lies in the area of computational geometry, finding and computing a good measure of convexity that satisfies the tenets of a metric: invariant under size and comparable. The measure would ideally also vary uniformly in its range as the nodes vary from 'bad' to 'good', and be discriminating enough that there are not too many border cases. Several different measures were considered and tried: skewness, corrected perimeter area, iso-perimetric quotient, and normalized mass-moment of inertia.

2.1.1 Skewness

Since the problem is restricted to a Cartesian plane, the first considered method for determining a convexity was comparing a modification of the third moments (skewness) at the extrema of the shape

$$S_x = \frac{(x_m - \mu)^3 + (x_M - \mu)^3}{A}$$

$$S_y = \frac{(y_m - \mu)^3 + (y_M - \mu)^3}{A}$$

of each node along the x and y axes, looking at the extrema on each axis. These extrema are denoted x_m for the minimum value along the x-axis, and x_M for the maximum value along the x-axis, similarly for y. μ here is the relevant x or x coordinate of the centroid (the average of all x values and y values of the cluster), and A is the total area of the shape. This had the advantage of leveraging the native way that Python stores its ordered pairs for a fast computation time. However, the information was only of limited use; although it gave a measure of how relatively skewed the cluster was, it was not comparable across clusters as skewness is proportional to the area of the cluster. As such, a small cluster that would be relatively good would be classified as bad, and a large cluster that is relatively bad would be classified as good. This measure also had issues with identifying non-convexity in symmetrical clusters, the easiest example being a cluster in the shape of an X or +, which would give $S_y = S_x = 0$ despite obviously not being a convex shape. Further research into the area of shape classification yielded three more computed measurements.

2.1.2 Corrected Perimeter-Area

The second measure considered was the corrected perimeter-area ratio (CPA), defined as

$$CPA = \frac{\alpha P}{\sqrt{A}}, \alpha = \frac{1}{2\sqrt{\pi}}$$

where P is perimeter and A area [3]. This measurement has the property that $CPA = 1$ for a perfect circle, then tends to infinity as the shape grows progressively elongated. During

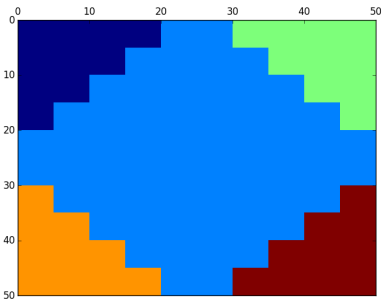

Figure 2.1: CPA example 1

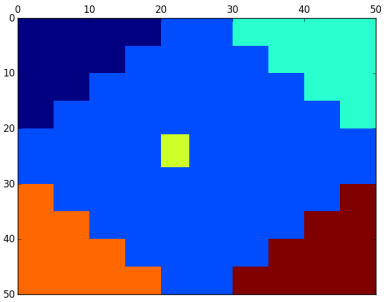

Figure 2.2: CPA Example 2

testing on randomly generated data it was found that it was very rare to see $CPA > 3$, and this data did include some very snake like and overall 'bad' clusters. But, the difference between clusters with $CPA < 2$ was not readily reflected in the measure, the observed difference between a score of 1.4 and 1.5 could be very large. This proved useful in identifying truly horrendous cases using $CPA > 2$ as a cutoff, but the CPA did not provide the fine discrimination that was needed for what were, by inspection, considered border cases. A comparison between Figure 2.1 and Figure 2.2 utilizing the light blue diamond in the center is in table 2.1. Note that the small yellow square inclusion worsens the CPA score by 0.15, a rather hefty adjustment. Also, note As such, the corrected perimeter-area measure is useful for quick identification of non-convex clusters, but proved unsuitable as an optimization objective criteria.

Table 2.1: Light blue area measures for Fig 2.1 and Fig 2.2

Measure	Fig 2.1	Fig 2.2
CPA	1.311	1.468
IPQ	0.582	0.464
NMI	0.956	0.926

Table 2.2: Yellow square measures for Fig 2.2

Measure	Value
CPA	0.921
IPQ	1.178
NMi	0.917

2.1.3 Iso-perimetric quotient

The third metric of consideration was the iso-perimetric quotient [4], defined as

$$IPQ = \frac{4\pi A}{P^2}.$$

The IPQ takes values in $(0, 1]$. Also of note is the relationship between the IPQ and CPA, namely

$$IPQ = \left(\frac{1}{CPA}\right)^2.$$

As such, the behavior of the IPQ was expected to mirror that of the CPA, and it did to a large extent. The IPQ, like the CPA, suffered from some numerical stability issues when dealing with very small clusters giving scores as high as 6. We see an example of this behavior in Table 2.2, where both the CPA and IPQ scores fall out of bounds due to the small area of the yellow cluster. The IPQ measure fixed some of the problems with the CPA in that it allowed fine discrimination between clusters, but was found that it penalized too harshly for inclusions. In some sense, it is the opposite of the CPA as expected; if the IPQ says it is a good shape, it is a good shape. But the IPQ does not really give a good measure for the 'not good' shapes, the same way the CPA does not for the 'not bad' shapes. Ideally, a good metric would have ranges for 'good', 'ok,' and 'bad' rather than 'good' and 'not good' or vice versa.

2.1.4 Normalized Second Moment of Area

The fourth measure of consideration was first proposed by Li, Goodchild, & Church [6] and slightly modified by Li, Chen, et al in 2014 [5], borrowing an idea from classical mechanics, the second moment of area. This is also known as the mass moment of inertia and referred to as such in the aforementioned papers, but most physics sources seem to denote the moment of inertia as

$$I = \int_Q r^2 dm$$

i.e. the second moment of mass with respect to the distance from an axis r, rather than the second moment of area which seems to be with respect to the centroid of the area. The measure compares the second area moment of the shape with the second area moment of some reference object, (for continuity with references, this measure will be denoted NMI)

$$NMI = \frac{I_0}{I}$$

where I is the second area moment of the cluster and I_0 is the second area moment of the reference shape. These are both calculated with respect to the centroid of the shape. Three different shapes were considered, square, circle, and regular hexagon. Square and hexagonal shapes were considered since they are both completely packable, that is an area can be completely partitioned into a set of squares or hexagons of the same size with the minimal possible space left over. Such a structure is called a regular tiling (or tessellation) of the plane. Only three such regular tessellations exist, two being square and hexagonal and the other being triangular. Triangles were not considered since the original data was quadrille, and decomposing a square into a regular (equilateral) triangle is not a trivial process. Circles were considered since they are the simplest absolutely convex shape. The key relationship that makes this measure work is that reference shape has the same area as the cluster whose convexity we are interested in measuring. A modified variant called the normalized Mass Moment of Inertia was proposed by Li, Chen, et al [5] which attempts to account for uneven 'mass' distribution in the cluster, i.e. inclusions of a different land type. But since the goal is to have uniform clusters, the unmodified NMI measure was used instead. This measure

proved an ideal objective value for the convexity criterion for determining border cases since it is measuring the clusters mass distribution against a circle of the same area. Since it was decided that the clusters should be more circular than not, a threshold cutoff of $NMI \geq 0.6$ was decided upon to classify as 'good enough' for clusters whose CPA and IPQ measures were deemed 'bad.' It also is that case that, experimentally speaking, a good CPA or IPQ $\implies$ good NMI.

2.2 Graph Theory

Within graph theory, research was done primarily into graph generation and weighting drawing some inspiration from partial differential equations, as the goal is to approximate a diffusion process along the network. Breaking up the data into convex clusters allows for easy computation of the diffusion on each cluster. That process needs to be translated into diffusion over the whole area. As such, research into graph theory and some of its methods was a logical step. Some various network flow algorithms were considered, but a Monte Carlo random walk simulation was settled upon for its ease of use.

2.2.1 Creating the Graph

Creating the graph was done by taking the length between the centroids of any two clusters, then connecting those that are less than a specified length. This length will be dependent on parameters of what is trying to be modeled, roughly speaking that distance parameter should be the absolute furthest any one object of interest can move in one time step. Then, the distance matrix is modified again by taking each distance and subtracting the minimum radii for each of the two clusters.

$$d_{i,j} = \| z_i - z_j \| - (r_i + r_j)$$

where $\| \cdot \|$ is the Euclidean norm, z_i is the centroid of cluster i, and r_i is the minimum radius of cluster i. The 'minimum radius' r_i is defined here as the radius of the largest circle that can be inscribed within cluster i such that all points contained in the circle lay

fully within the cluster. This was done to make the distance attempt to reflect the actual 'border distance' between two clusters. Having to go through an exhaustive search through the border of each cluster to find the minimum distance to the border of each neighboring cluster would be an extremely slow and arduous process, which could be possibly alleviated with a clever parallel implementation for the searches. This however lays beyond the scope of this thesis.

2.2.2 Assigning Weights and Probabilities

Since the goal is to approximate the diffusion/heat equation on a graph, the weights of each edge should mimic the probability of a particle unit moving between any two clusters. As such the weighting function was drawn from the fundamental solution to the heat equation

$$\Phi(x,t) = \frac{1}{4\pi kt} exp\left(-\frac{x^2}{4kt}\right)$$

taking in particular the exponential part

$$exp\left(-\frac{x^2}{4kt}\right)$$

replacing x with $d_{i,j}$ and $4k$ with a scalar β. This yields

$$\omega_{i,j} = exp\left(-d_{i,j}^2 \frac{\Delta t}{\beta}\right).$$

β now becomes a diffusivity parameter that we can tweak to get the model to approximate real data, and Δt is a time-step parameter that is calculated for each iteration to ensure that the sum of all the probabilities leaving node i is less than one. Several other weighting functions were considered before deciding on the chosen function. These were mainly simple transforms such as $1/d_{i,j}$, $1/d_{i,j}^2$, and $\sqrt{d_{i,j}}$. Of these, $1/d_{i,j}^2$ seemed to work the best, generating reasonable weights. This follows, since from a physics point of view treating the clusters as point masses with a 'gravitational' type interaction the expected fall off in influence would be about the inverse of the square of the distance. However, none of these are as flexible as the final choice. These weights are then used in the calculation of

movement probabilities from node to node. The probability for movement from one node to another (assume that all movement is from node i to node j) should be proportional to four things: the distance between nodes ($\omega_{i,j}$), the number of ways to leave the node ($\|E_i\|$), the relative sizes between the two nodes ($\frac{A_i}{A_j}$), and optionally the 'agreeableness' of the destination node (F_j). 'Agreeableness' is a parameter in the range $[0-1]$ that tells how favorable the environment of a node is for the variable in question. The F values are used in a similar manner to the area ratios, since movement to a 'bad' area should be penalized but movement from a bad area should be encouraged. This gives a final equation of

$$P_{i,j} = \omega_{i,j} \frac{1}{\|E_i\|} \frac{A_i}{A_j} \frac{F_j}{F_i}$$

for each edge (i, j).

Chapter 3

Implementation

3.1 Python

The program for the project was written in Python 3 utilizing the NumPy, NetworkX, and Matplotlib libraries as needed. NumPy was used for its matrix processing abilities and probability functions, NetworkX for its graph generation and plotting methods, and Matplotlib for its plotting library. The initial program consists of two parts, a class and method definition file (cluster.py), and a driver file (driver.py) that calls the procedures. At the moment, driver.py reads a CSV file in that is specified when the program is called on the command line. There is also a global parameters file (cluster_parameters.py) that allows for easy control of the important variables that have high impact on the data. These include minimum allowable cluster size, maximum center distance, maximum number of decompositions per cluster, as well as important functions such as the weighting function, norm function, and probability function. A more thorough explanation is in Appendix B. The whole method consists of four distinct procedures: clustering, decomposition, graph creation, and simulation.

3.1.1 Clustering

The command line specified CSV file should contain the raw pixel data, which is read into a NumPy matrix. The scale of the data is accounted in the global parameters file, in

particular within the various 'distance' parameters. Then the clustering procedure can run. A secondary matrix of the same shape as the data matrix is created to keep track of cell visitations. The data and visitation matrix are both traversed left to right, finding the first non visited cell, then running a breadth first search routine from that cell to collect all contiguous cells that share the initial data value. This is repeated until the entire raw data array has been traversed, and every cell belongs to a cluster.

3.1.2 Decomposition

Decomposition involves traversing each cluster that fails the NMI criteria and performing two sweeps, one horizontal and one vertical, across the cluster. During each sweep, every line segment that connects two border points along the respective axis and is also contained within the cluster is appended to a list. These are then sorted in order of length, and for each entry on the list the cluster is split along that segment, then re-collected into two different clusters. The relative change in the NMI is then measured by first computing the NMI ratio compared to the original NMI for both new clusters.

$$\hat{R}_1 = \frac{I_1}{I} \qquad \hat{R}_2 = \frac{I_2}{I}$$

Then, in order to compare and choose the 'best' split, the geometric mean of the two ratios is computed

$$R = \sqrt{\hat{R}_1 \hat{R}_2}$$

and the maximum of all means is chosen. To prevent some degenerate cases, there is also a parameter that controls the maximum number of times that a cluster can be split. The program's behavior in these cases usually devolves into splitting off large square pieces from the initial cluster, since the NMI of a square is approximately 0.95 it will consistently be one of the higher ranking candidates.

3.1.3 Graph Creation

Once all of the offending clusters have been addressed, we now turn the cluster map into a graph. The initial adjacency matrix is created from all $d_{i,j}$ values. Any values which are above the specified distance threshold are then zeroed out. Then the remaining values have the weighting function applied, which is then used to create the probability adjacency matrix $P(\Delta t)$. This matrix is then iterated through with decreasing Δt values until a suitable value $\Delta\hat{t}$ is found such that each row sum is less or equal to 1, i.e.

$$\forall i, \sum_j P_{i,j} \leq 1$$

This ensures that the total movement probability is in fact, a probability.

3.1.4 Simulation

The current simulation setup is a basic Monte Carlo type, drawing numbers from NumPy's random binomial function using $n = 1, p = P_{i,j}$. Each cluster has a variable that tracks the number of times it is visited over the entire simulation. Movement is allowed from present to non-present and from present to present for the simulations that follow, but can be switched easily to movement only from present to non-present if needed. This was chosen to try and determine if there any unwanted or strange behaviors from the decomposition process that might have an adverse affect on the simulation, such as favoring one particular set of cluster parameters too much (size, NMI, etc). Each simulation also starts with one 'infected' cluster in the top left corner at $(0, 0)$. Rather than simulating for a set period of time, a set number of time steps was specified. Since $\Delta\hat{t}$ is calculated differently for each set of data. For the data sets in this chapter, the random 80x80 data set had a calculated $\Delta\hat{t} = 5.8$ and the 500x500 Arkansas data set had a calculated $\Delta\hat{t} = 10.01$

3.2 Examples

Three sets of data were used for the method, a set of predetermined shapes, a set of randomly generated data, and real geographic data from Arkansas. A set of ten predetermined shapes were created to test the feasibility and characteristics of the decomposition algorithm, trying to determine if there were any quirks or pathologically bad cases that the algorithm would fail upon. The random data was generated from a random matrix of ones and zeros, then expanded by a factor of ten to give a decently sized data set. Three sets were generated, $80x80, 250x250$, and$1000x1000$. The figures here are drawn from the $80x80$ set, for ease of visibility. See Appendix A for the other two generated datasets. The Arkansas data is a subset of the National Land Cover Database data for Arkansas at $500x500$ part of the northwest corner near Hiwasse/Gravette Arkansas, $1000x1000$ in the northern part around Hardy/Cherokee Village, and $2000x2000$ which encompasses the Greater Little Rock area. The resolution for this data is rather fine, coming in at $30m^2$ for each square. The NLCD [1] has the values for land cover detailed in Figure 3.9

3.2.1 Predefined shapes

These shapes were created to see how the split algorithm would deal with irregular shapes. Ten different objects were designed, some to fail and some to succeed. Three that were designed to be operated upon have been included; Shape 1 is detailed here, and the other two, Shape 3 and Shape 9, are detailed in Appendix A (A.1.1, A.1.2). The parameters were used for the predefined datasets are in Table 3.1.

Table 3.1: Parameters for predefined figures

Parameter name	Value
NMI threshold	0.6
Distance threshold	50
Number of splits	6
Minimum cluster size	40

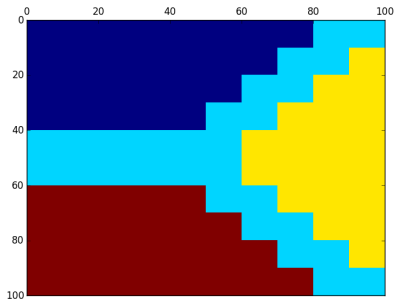

Figure 3.1: Shape 1

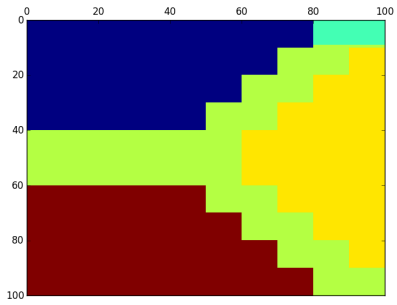

Figure 3.2: Shape 1 after one split

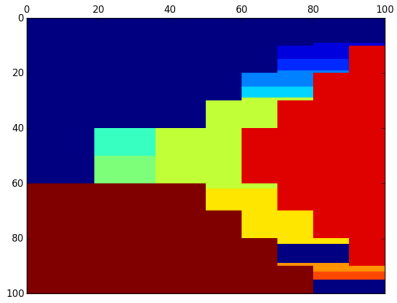

Figure 3.3: Shape 1 after six splits

In Figure 3.1 we see the first shape that was tested after the initial clustering. Each color is a distinct cluster. Figure 3.2 shows the layout of the clusters after one iteration of the decomposition algorithm. Note the rectangle on the upper right. Figure 3.3 shows the layout after 6 iterations of the decomposition algorithm. These last two show a good example of the algorithm's terminal behavior. Since the upper right rectangle in Figure

3.2 has $NMI \geq 0.95$, it gives the highest possible ratio increase which is picked up by the geometric mean. This is repeated in the rest of the tests; when possible, the algorithm carves off a square or rectangular shaped cluster from one of the ends. We see this clearly in Figure 3.3 with the preponderance of small rectangles.

3.2.2 Example with random data

The random data was generated as a binary data matrix (0 or 1) at a small size, then each block was expanded by a factor of 5. This was done to ensure sufficiently large cluster sizes to test. Table 3.2 gives the parameters for the random data tests. Each data set was run with the same set of parameters.

Table 3.2: Parameters for random 80x80 matrix

Parameter	Value
NMI threshold	0.6
Distance threshold	20
β	4π
Number of splits	2
Minimum cluster size	40
Time steps	50
Number of simulations	10000

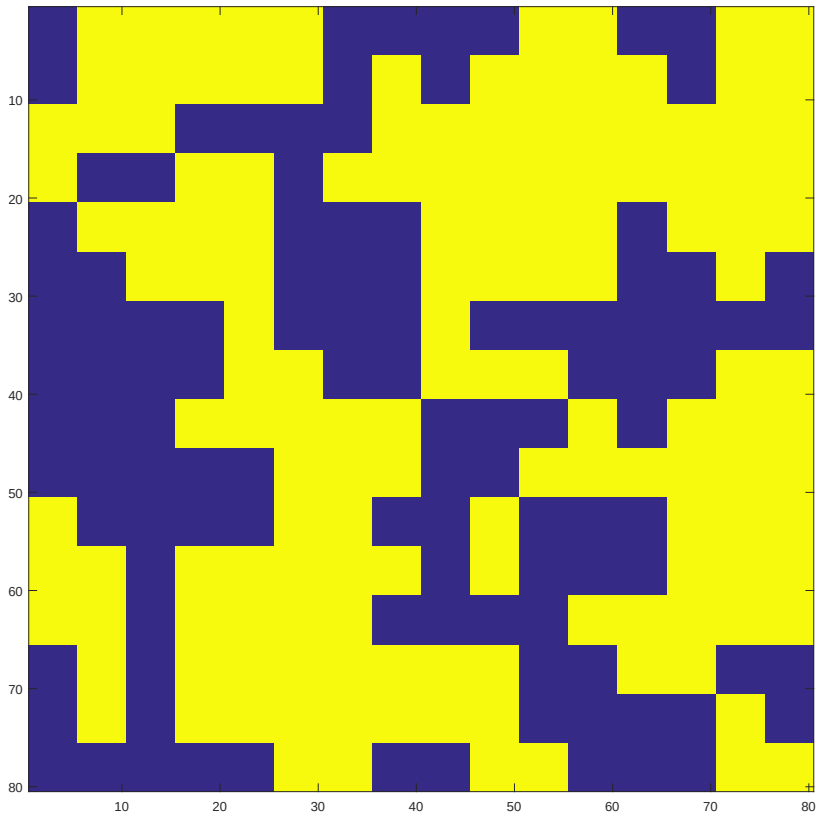

Figure 3.4: Random data 80x80

Figure 3.4 shows the initial layout of the data before decomposition has taken place. Of special note is that there are several large irregularly shaped objects on the left that will be good candidates for the splitting algorithm.

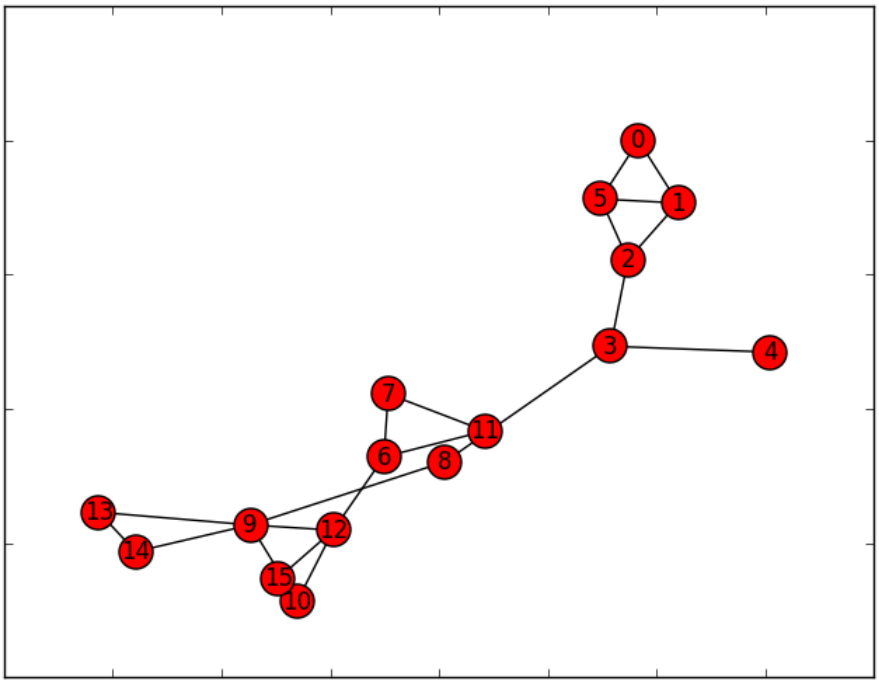

Figure 3.5: Random data before graph

Here Figure 3.5 shows the layout of the data when drawn as a graph. We see the effects of having large, irregular clusters as the graph admits only one possible path across the graph. This would be fairly uninteresting to study and the conclusion obvious, remove Cluster 3 to prevent spread. However, Cluster 3 is rather large, thus possibly impractical to 'remove.'

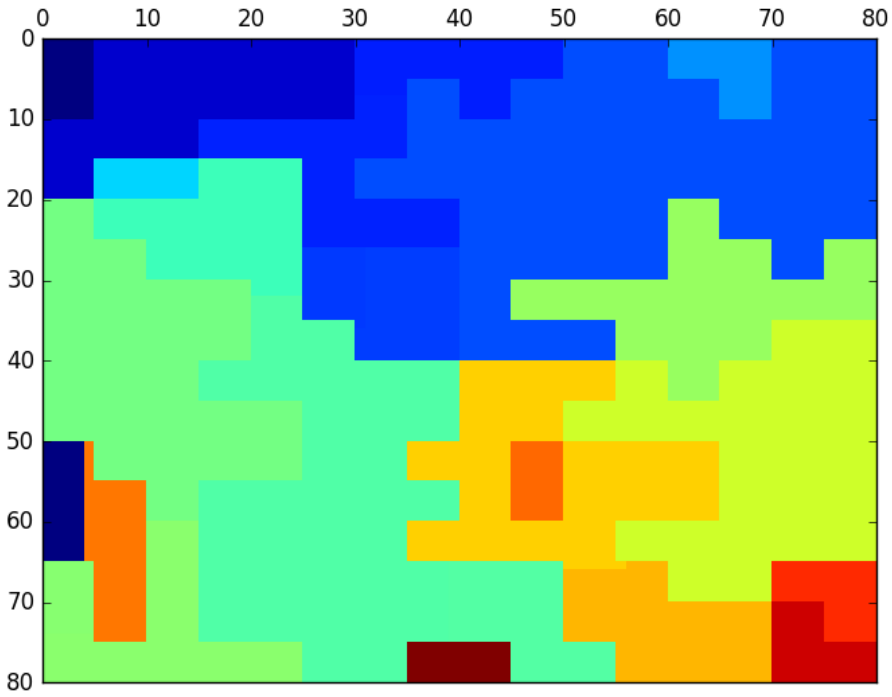

Figure 3.6: Random data plot after decomposition

Figure 3.6 shows the data after one split sweep. Comparing to Figure 3.4, several of the rather 'bad' shapes have been split into better ones.

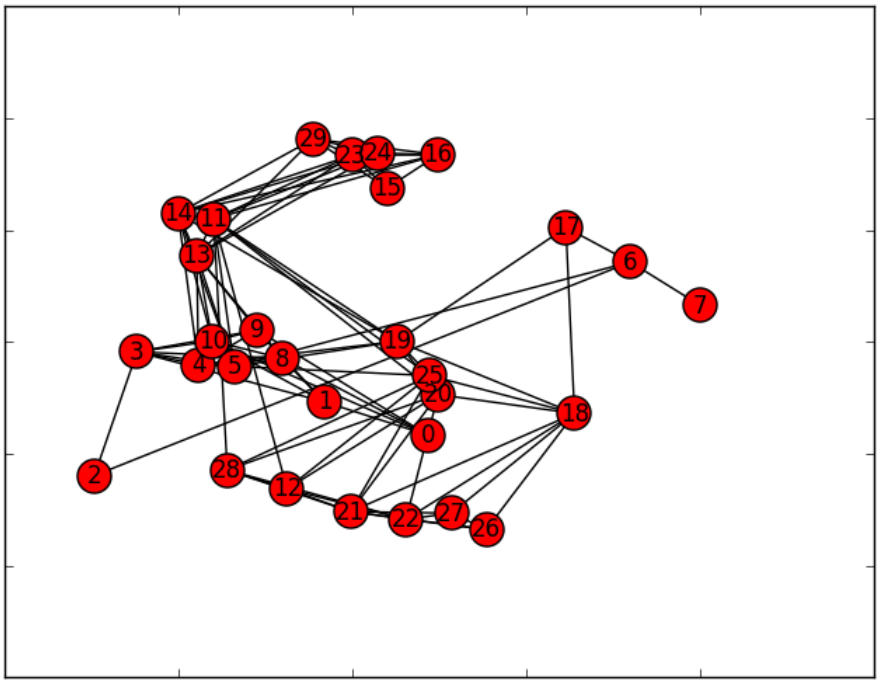

Figure 3.7: Random data graph after decomposition

Figure 3.7 shows a much more interesting graph after the one decomposition sweep compared to Figure 3.5.

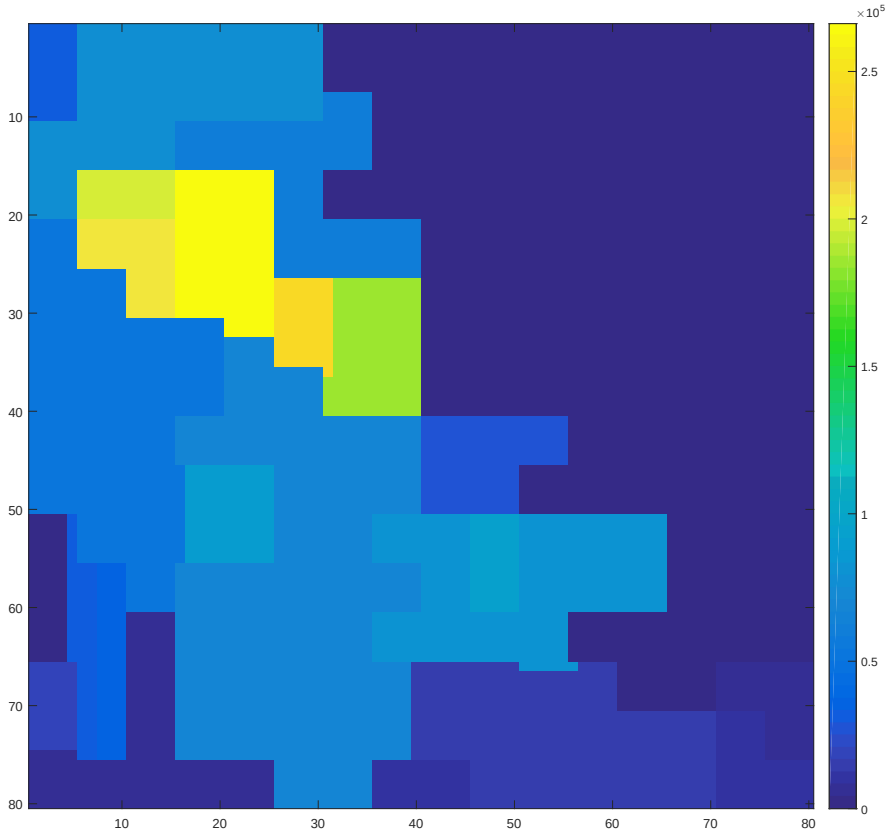

Figure 3.8: Random data clusters visited after simulation

Figure 3.8 shows the results of a simulation on the decomposed data set. As it is set, the simulation seems to favor same size clusters a bit too heavily, as the yellow clusters in the middle were visited far more times than the total number of times the simulation was run. The simulation was run 10,000 times, and the yellow clusters were visited about 250,000 times. This means that the current formulation for the probability incentivizes movement to the same area too greatly. Using the logarithm or square root of the area ratio would perhaps provide a damping effect that would allow for a more even spread.

3.3 Application

With the two test scenarios done, now the method is applied to actual data. Three different data subsets were taken from the NLCD data of the state of Arkansas, a $500x500$ set in the northwest corner near Gravette, a $1000x1000$ set in the north around Hardy, and a $2000x2000$ block surrounding Little Rock. Each of these three sets were chosen for a reason. The Gravette area has a mountainous forest transition into farmland blocked off by roads. The Hardy area has lots of small lakes in a hilly region that is primarily forests. Little Rock is a large urban area that has hills to the north, farmland to the south, and swampy wetlands to the east, with the areas being roughly partitioned by the Interstates running through the area.

This data is the underlying structure that is used to simulate the spread of invasive wild hogs. The values of the data are shown in figure 3.9. This classification was slightly more detailed than was wanted for validating the model as too many sub-categories could possibly dilute the cluster milieu too much. So each value was rounded down to the nearest multiple of ten leaving the major categories, as shown in Table 3.3.

Table 3.3: New NLCD Legend

Label	Meaning
10	Water
20	Developed
30	Barren
40	Forested
50	Shrubland
70	Grassland
80	Cultivated land
90	Wetland

In this way, sufficiently large clusters were assured. Also, to alleviate strain on the clustering system, when the method was applied to the Arkansas data only values 30 and above were considered. The primary purpose of this was to ignore roads, since long, spindly shapes are usually considered bad and have the highest likelihood of becoming a degenerate case. This is also why the water category is ignored as well. For just the smallest Arkansas data set, this reduced the number of clusters from over 6000 to just over 2000. Considering

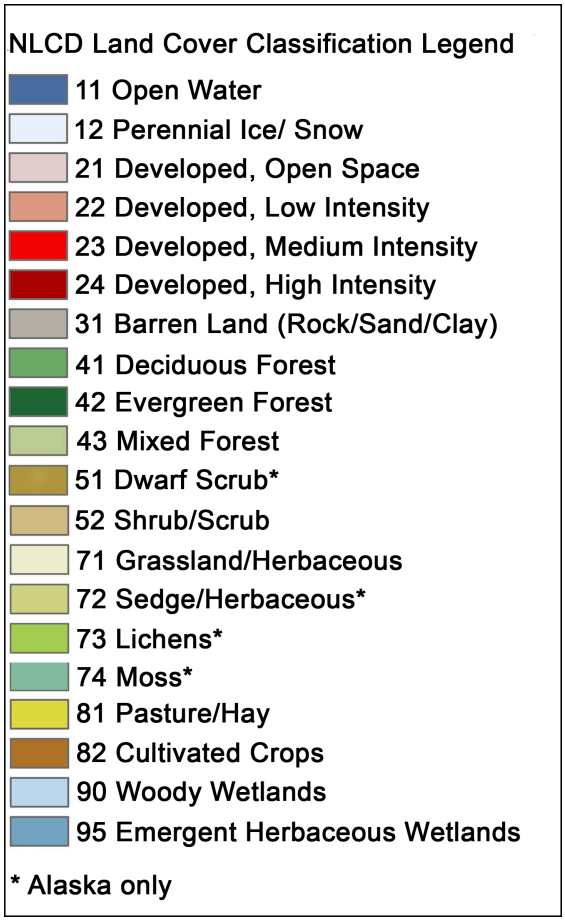

Figure 3.9: NLCD Legend

that there are only 10,000 original data points, this is a good reduction in the dimensionality of the data.

Table 3.4: Parameters for Arkansas 500x500 matrix

Parameter	Value
NMI threshold	0.6
Distance threshold	50
β	4π
Number of splits	1
Minimum cluster size	40
Time steps	50
Number of simulations	10000

All parameters were held the same from the random data, with the exception of the maximum distance and number of splits. These are shown in Table 3.4. Max distance was

increased to ensure that the adjacency matrix was not too sparse as to be uninteresting, and the number of splits was reduced to speed up computation time necessary.

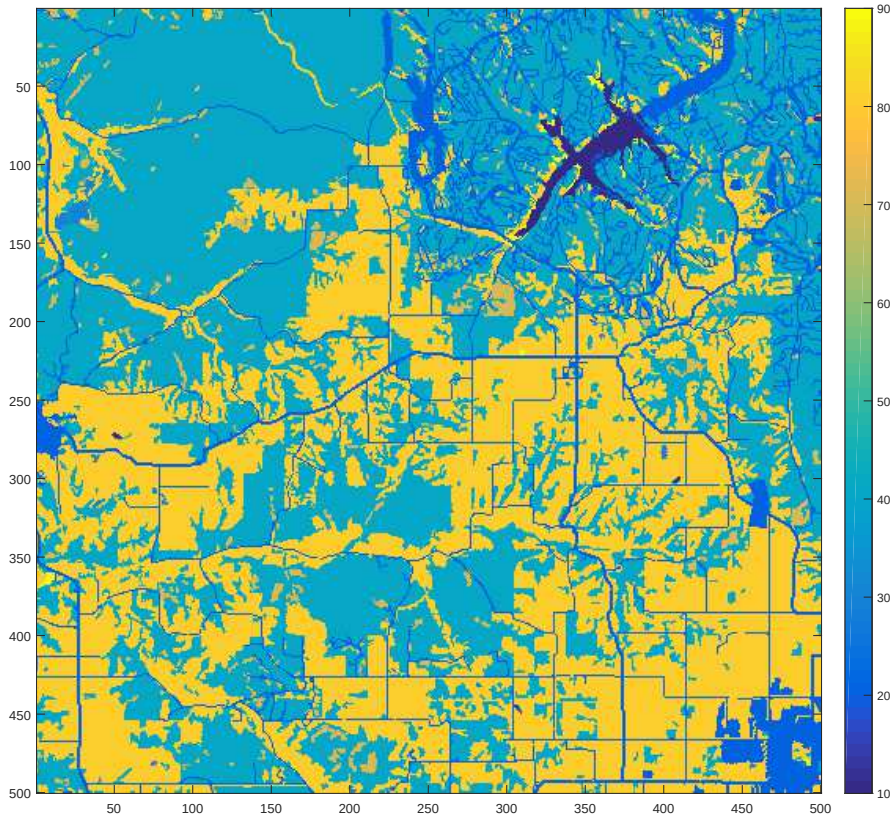

Figure 3.10: Arkansas Data

Figure 3.10 shows the raw values of the data. Roads and inhabited areas can clearly be seen.

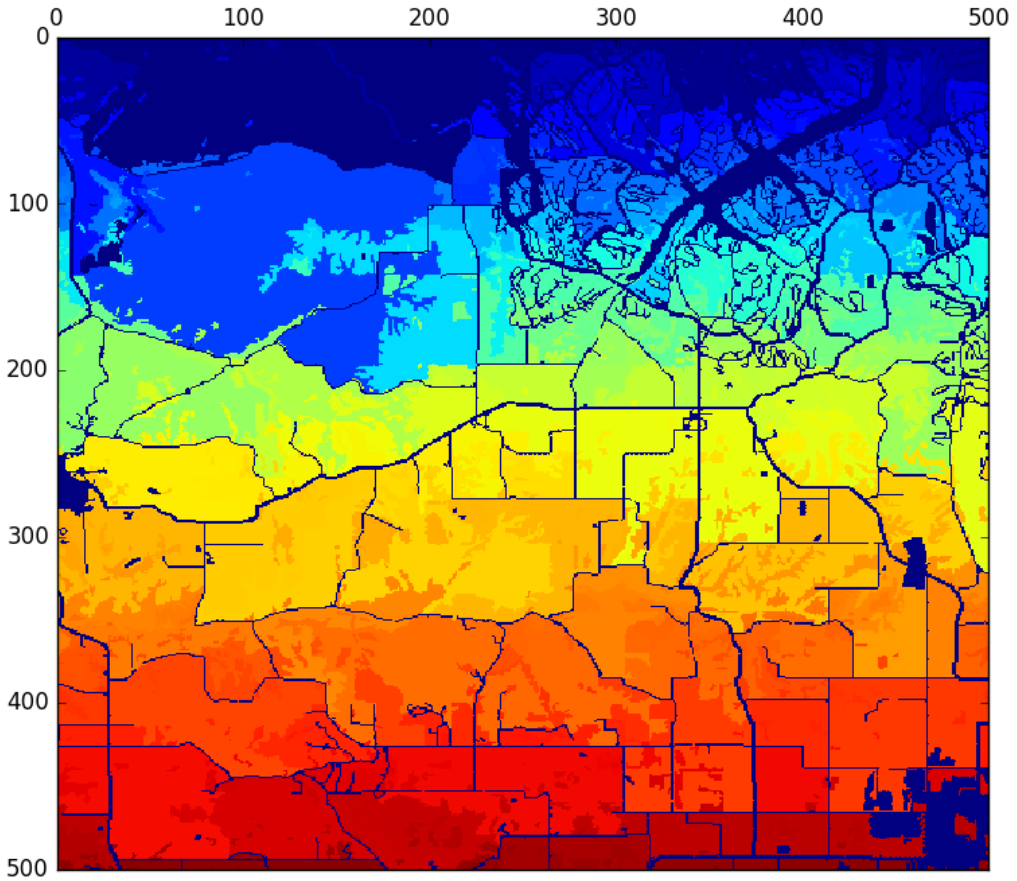

Figure 3.11: Arkansas Data after initial clustering

Figure 3.11 shows the same data after the initial clustering.

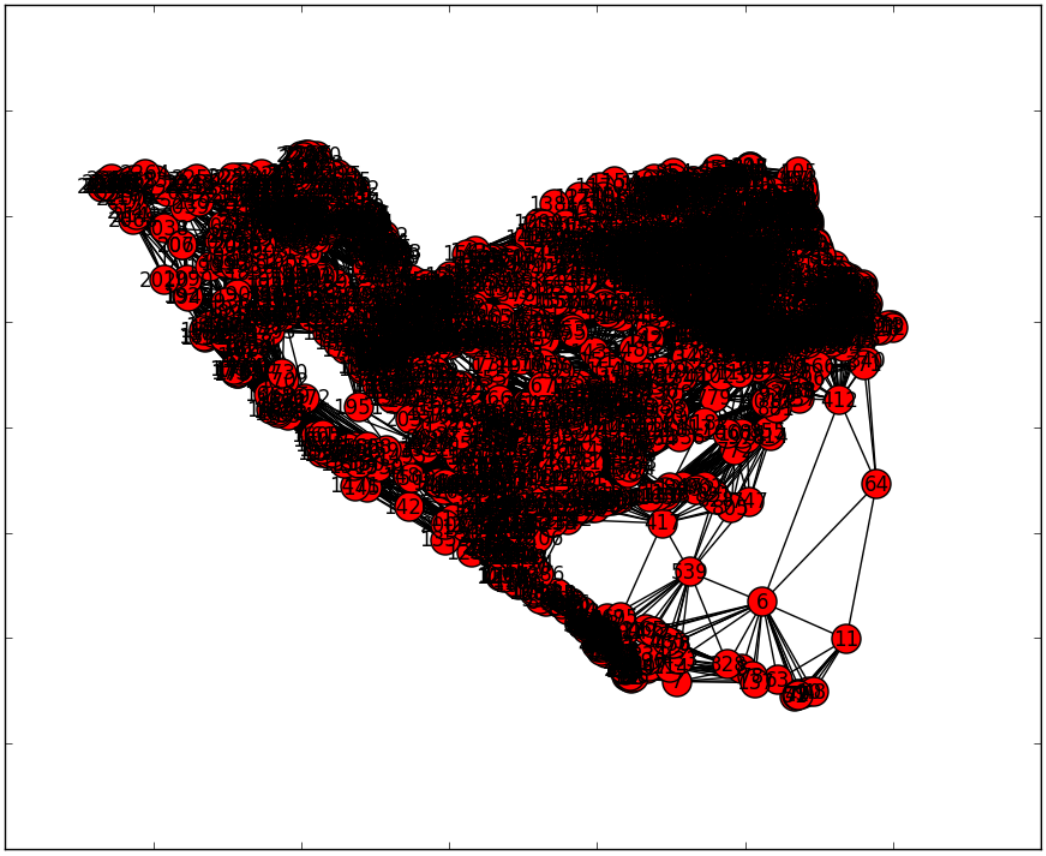

Figure 3.12: Arkansas Data initial graph

Figure 3.12 shows the initial graph generated from that clustering. The total number of clusters involved makes discerning much from the figure difficult. But some interesting clusters can be seen, such as clusters 6, 64, and 11 dominating their locality.

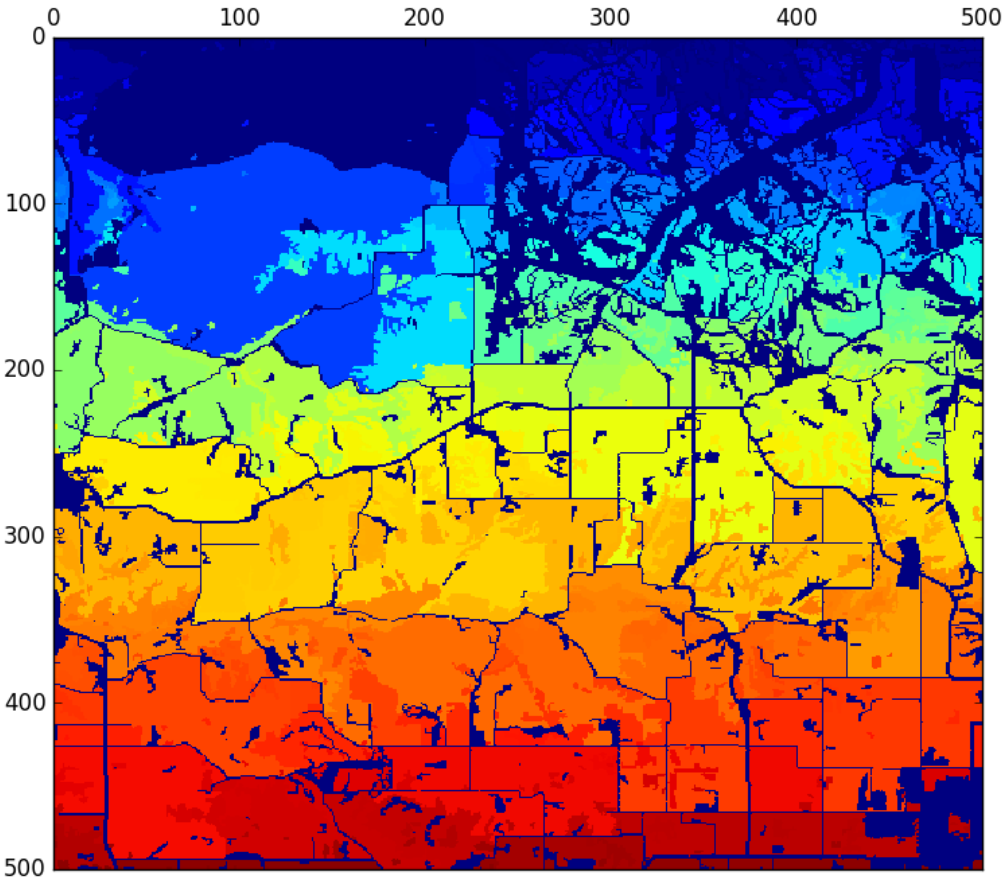

Figure 3.13: Arkansas Data after cluster decomposition

Note again the larger dark blue spots at the bottom areas. Some, like the roads and lakes, are because they were not considered in the first place, thus receive a label of 0. Others, like the irregular shapes in fields are there because the data somehow leaked out of the data structure.

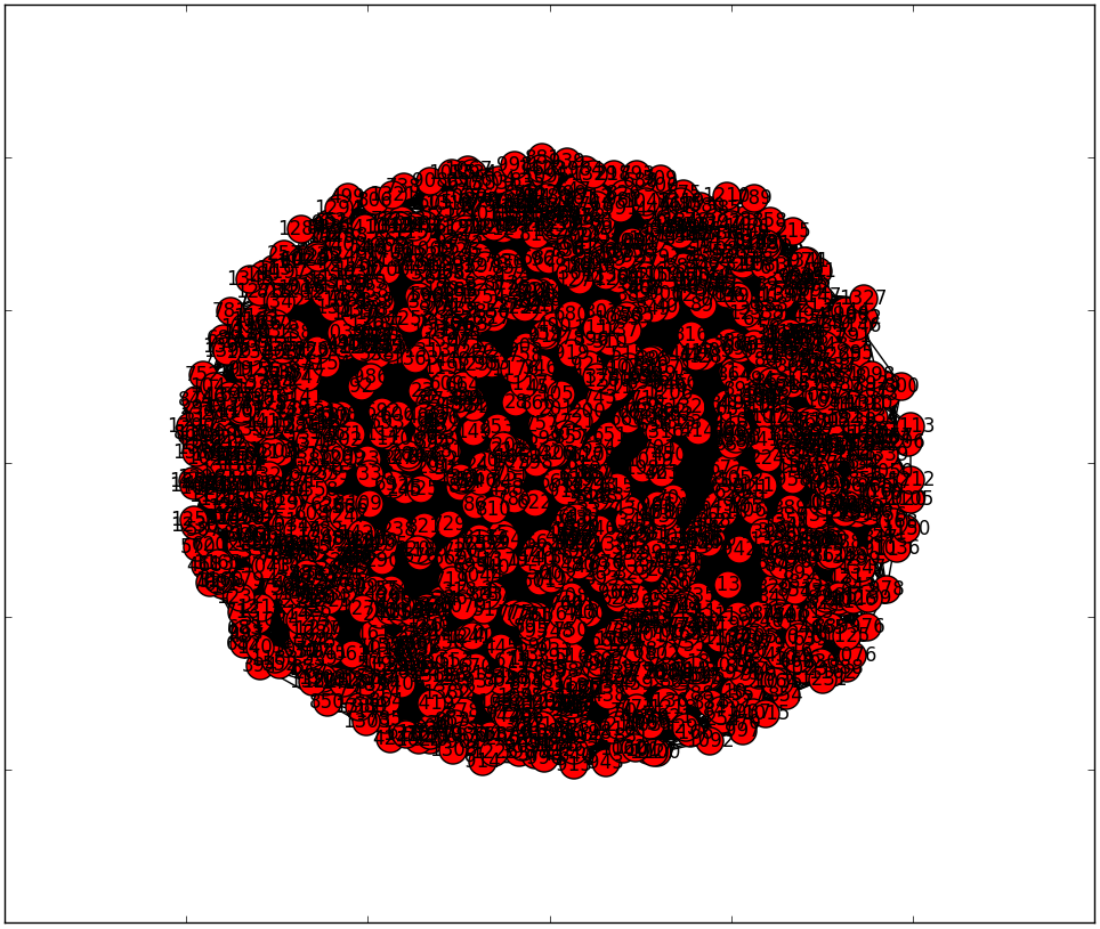

Figure 3.14: Arkansas Data graph layout after decomposition

Figure 3.14 shows the resulting graph after one decomposition sweep. Compared to Figure 3.12, a much more uniform structure is observed.

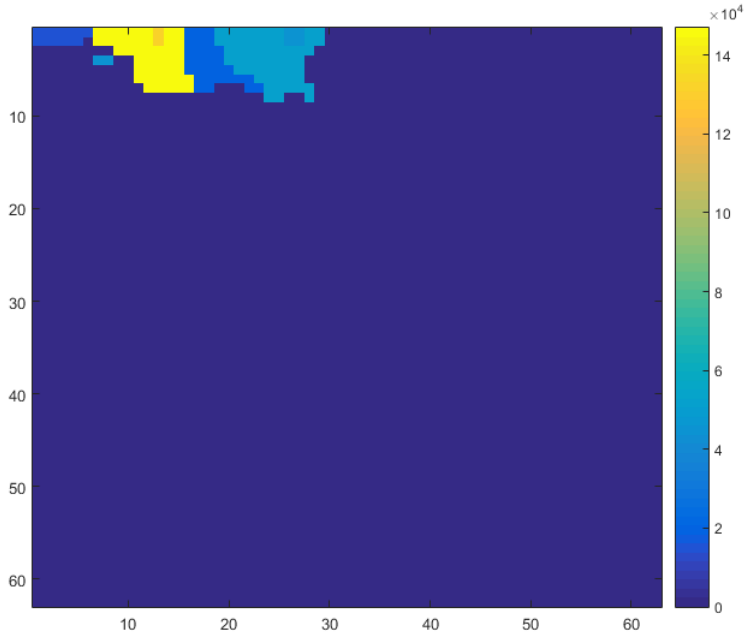

Figure 3.15: Arkansas Data zoomed to relevant visited area

Since the parameters were held constant, this graph shows that 50 time steps was not enough to get a good pattern of spread with the current probability function setup. Further tweaking of that function would be needed, as well as some comparison data, in order to generate a better result.

Chapter 4

Discussion

4.1 Summary

This thesis developed a methodology for systematically reducing the dimensionality of high resolution data problems in a manner that maximizes the convexity of its subsets. This allows for simplification of assumptions when a spread model is run on the data as non-convex subsets do not need to be handled. The method generates a usable data set that is roughly on the same order of magnitude (± 1 was observed in these data sets) as one dimension of the original data. This is a good reduction in the total number of data points; the $500x500$ Arkansas data started with 250,000 original points which the method reduced to about 2500 clusters.

4.2 Advantages

This method has some advantages when compared to a standard areal diffusion model, primarily concerning resource allocation questions. With areal diffusion, finding answers to the question which areas should be targeted to achieve a particular result, be it tracking, prevention of spread, or encouragement of growth, is not a particularly straight forward process. A two dimensional diffusion approach to this problem would involve assigning each type of data value (here, land cover type) a different density, then running the diffusion equation over the surface. This is very useful when looking for answers related to extant and

pattern of spread, but in order to identify which areas would change the pattern of spread the most involves looking at individual areas, modifying them, and running the model again. In this graph theory based method, these answers fall naturally out of the graph's structure once analyzed. For instance, using Kruskal's algorithm to find the maximal tree is an $\mathcal{O}(E \log V)$ operation, and analyzing the removal/modification of each node is at worst $\mathcal{O}(VE \log V)$ and the removal/modification of each edge $\mathcal{O}(E^2 \log V)$. So, the worst case complexity for analyzing the entire graph is $\mathcal{O}(VE^2 \log V)$. In practice it would probably be smaller, since the generated graphs are usually far from complete, with $E \sim nV$ for some $n < V$ rather than $E = V^2$.

The other advantage of this method is it lends itself naturally to a parallelization scheme in all three methods, breadth first search, the convexity analysis, and the simulation. The convexity analysis of each cluster is an independent operation, requiring only the data in each cluster to compute and make a decision for the cluster. The BFS routine only needs access to the original data matrix, visited matrix, and a starting coordinate. This would not be as fully parallelized as the convexity analysis, but would still allow for a substantial speed increase. But, the Monte Carlo simulation the prime target for parallel acceleration. Since the larger data set needed more time steps to generate interesting data, the simulation procedure became the biggest bottleneck in the method. Increasing the number of time steps from 50 to 500 increased computation time tenfold, from about 3 seconds per run to about 30 seconds per run. Correspondingly, total computation time went from around 8 hours to 87 hours. Parallelizing this particular method, even if it only sped up by a factor of 4, would bring the total time to under a day.

4.3 Limitations and Future Work

4.3.1 Limitations

There are a few limitations with this particular implementation, in both the computational and mathematical senses. Computationally, memory management is fairly poor as 8GB was not enough to run the larger Arkansas data models do to the structure of the data.

The random data managed to run in a reasonable time (less than an hour) for the two smaller sizes, but the two smaller Arkansas data sets took about 8 and 14 hours respectively apiece. The two large datasets however did not finish quickly at all, with their computational time measured in days. The $2000x2000$ data set around Little Rock in particular did not terminate within a week of execution, the raw data is in Appendix A for completeness. Similar, the $1000x1000$ randomly generated data did not terminate, though the initial plots were generated. The current program version also has a bug that has not been tracked down, that for lack of a better term can be called 'data leakage.' Simply, not all the clusters are stored after the decomposition, resulting in some areas not being considered during the simulation. The number of clusters that go missing is related to the total number of splits allowed, but of no discernible function. For example, the random $80x80$ data set loses no clusters after the first split sweep, loses one after the second, adds another four after the third, another two after the fourth, and only one after the fifth, but after six sweeps the maximum number lost is eight, with no more being lost even up to ten total sweeps. The other two randomly generated data sets exhibited similar behaviors, only losing a certain amount of clusters total, no matter the number of sweeps. It is for this reason that the number of sweeps was capped at two for this thesis, and usually reduced to one, to minimize the effect of this unexplained behavior. This is a limitation of the current implementation in Python due to the way the language handles its built-in data structure types and the manipulation thereof. An implementation in C/C++ would not only be somewhat faster, but the way it handles its data structures is more simplistic in nature and would likely preclude an error of this type occurring.

4.3.2 Future Work

For future work, several extra procedures can be created. First, tweaking the method so that it also looks at the average global convexity would be a useful addition. This would be an additional way of looking at the convexity measure of the data set; instead of specifying a minimum convexity for each cluster the user could specify that the average convexity of all the clusters needs to be above a certain threshold. Another addition would be a procedure to generate the parameters of the simulation from some existing parent data. The parameter

matching optimization procedure would be of great benefit for modifying the probability weighting function, allowing the user to find meaningful variables for the function.

The data leakage bug needs to be tracked down and dealt with. Perhaps by porting the code over to another language, MATLAB or C++, the error would become more obvious. After that, the program could use some optimization in terms of speed, taking advantage of its current structure. While trying to eliminate the leakage problem, the program was intentionally rewritten in a very safe way, with data being copied, manipulated, then copied back with no in place operations. Eliminating these deep copies would go a long way in optimizing the program. As mentioned before, program was also intentionally written to be accelerated and parallelized as much as possible. Implementing the computationally intensive parts of the program, particularly the split operation and Monte Carlo simulation, with NVIDIA's CUDA or the OpenCL API would be greatly beneficial.

Mathematically, other types of sampling should also be tried other than binomial type, perhaps treating the movement as a Poisson type process. Restructuring the graph generation into only looking at strictly adjacent clusters would be interesting as well generating an even sparser adjacency matrix, perhaps allowing movement to be treated as a Markov chain type process where there is a chance of staying in the cluster. This would allow application to migration or nomadic type movement patterns, where transfer from one cluster to another would be dependent upon time spent in the current area, which itself is related to resource availability.

In terms of structure, the cluster splitting algorithm needs to be improved, as there has been shown several cases where the current version just cannot cope with the layout of the data. In particular, the current version has real problems with dealing with pseudo-symmetric inclusions. Since the 'test boundary' currently works only with line segments between adjacent border points; if any segment of the line that runs between two outside border points crosses an inclusion, searching on either side of that segment both yield the entire cluster. A pathological example is shown in Figure 4.1.

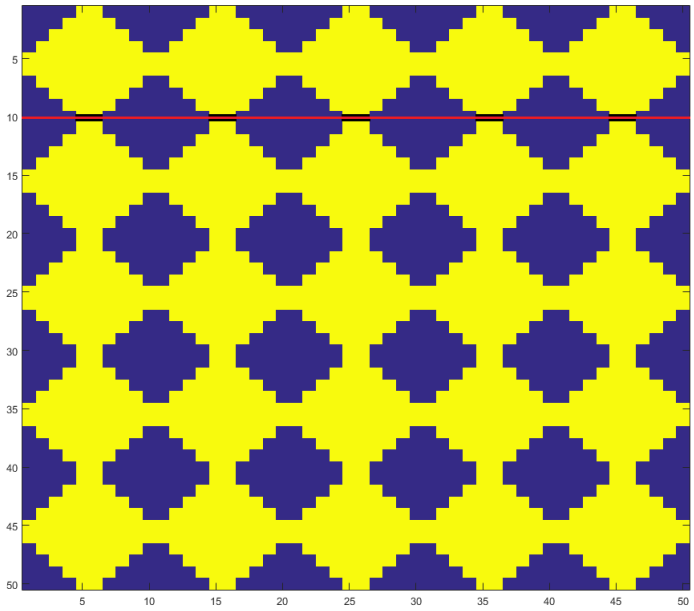

Figure 4.1: Pathological example

If the red line is the current split axis, where it is highlighted in black denotes the segments that would be fed to the search routine in the split procedure, one at a time. Note that both sides would yield the entirety of the yellow cluster when the search routine is called on the side in question. Perhaps, once this case has been identified, the fall back would be to split along the entire red line, then call the search routine. Once these issues are rectified, and some further testing carried out, this method should be a useful tool when looking at resource allocation optimization questions.

Bibliography

[1] National land cover database 2006 product legend.

[2] Joseph Corn Graham Hickling Marguerite Madden Agricola Odoi Charles Collins, Eric Carr. Modeling spread of an invasive vertebrate pest: feral swine in arkansas, usa.

[3] Almo Farina. *Principles and Methods in Landscape Ecology: Towards a Science of Landscape*, volume 3. Springer, 2 edition, 2006.

[4] Robert Osserman. The isoperimetric inequality. *Bull. Amer. Math. Soc.*, 84(6):1182–1238, 11 1978.

[5] Elizabeth Wentz Wenwen Li, Tingyong Chen and Chao Fan. Nmmi: A mass compactness measure for spatial pattern analysis of areal features. *Annals of the Association of American Geographers*, 104(6):1116–1133, September 2014.

[6] Michael Goodchild Wenwen Li and Richard Church. An efcient measure of compactness for two-dimensional shapes and its application in regionalization problems. *International Journal of Geographical Information Science*, 27(6):1227–1250, 2013.

Appendix

Appendix A

Additional Plots

A.1 Other pregenerated shapes

A.1.1 Shape 3

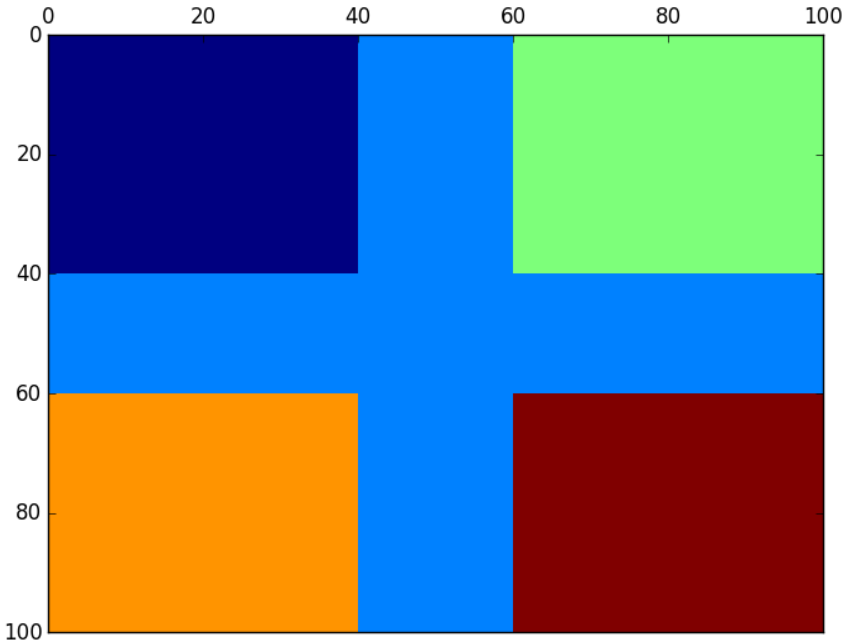

Figure A.1: Shape 3

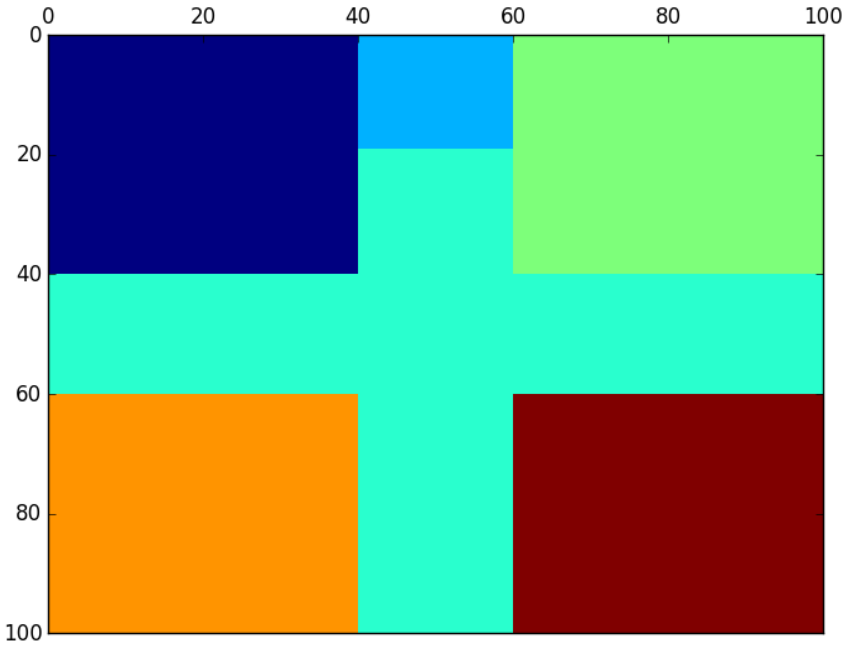

Figure A.2: Shape 3 after one split

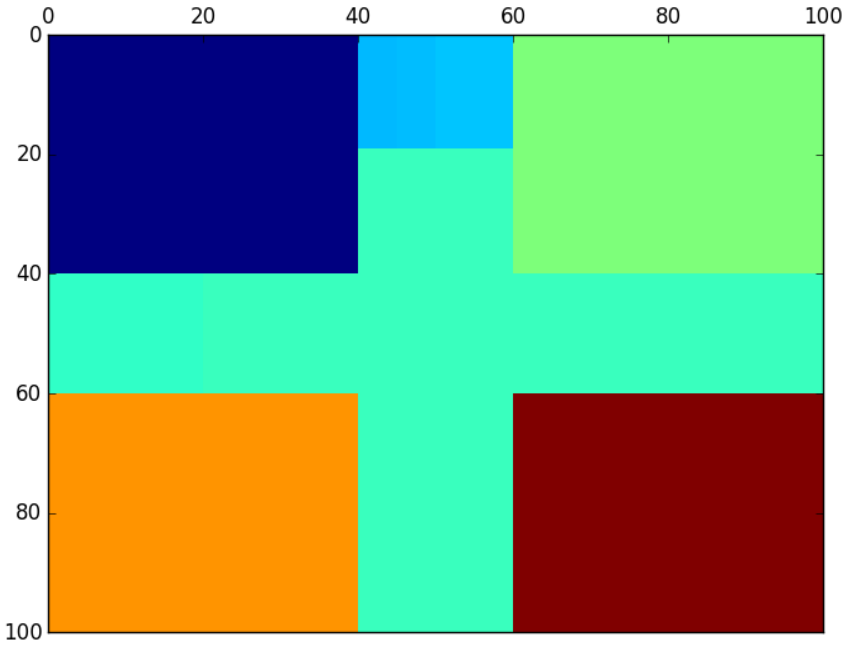

Figure A.3: Shape 3 after two splits

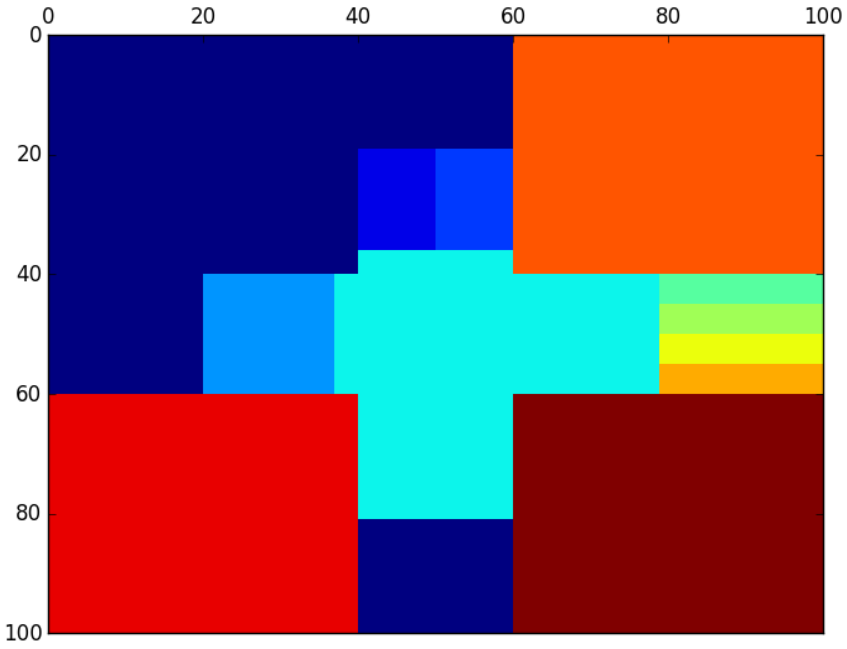

Figure A.4: Shape 3 after six splits

A.1.2 Shape 9

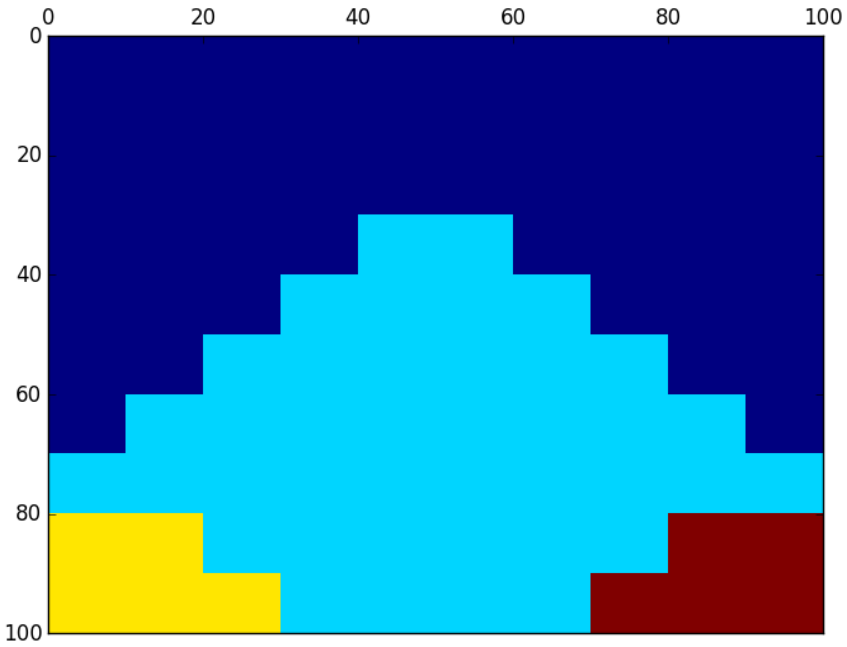

Figure A.5: Shape 9

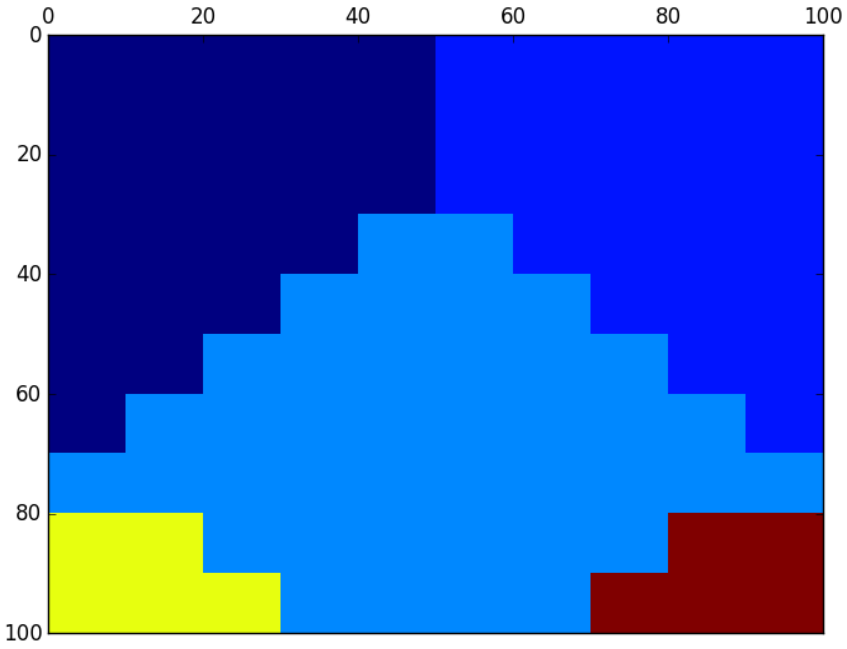

Figure A.6: Shape 9 after one split

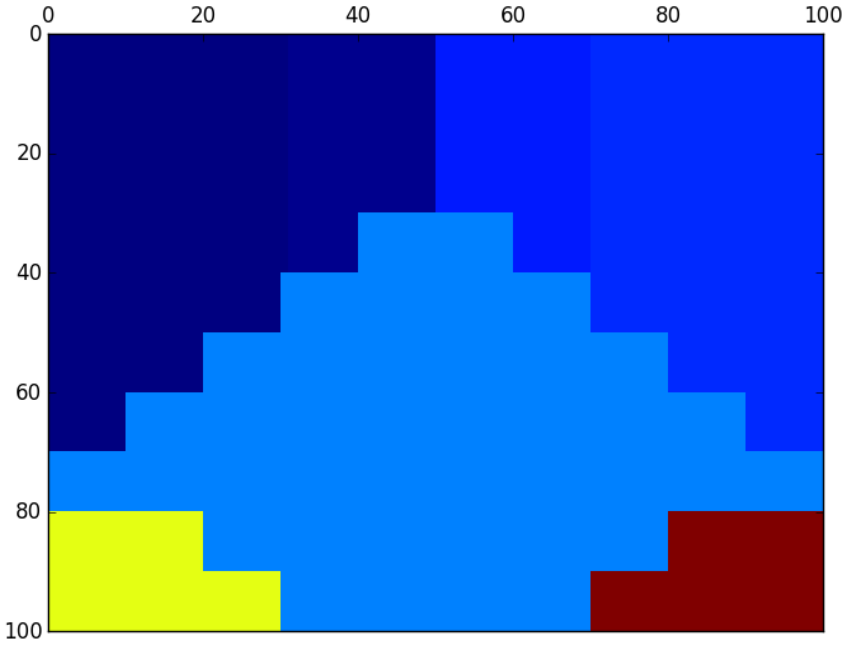

Figure A.7: Shape 9 after two splits

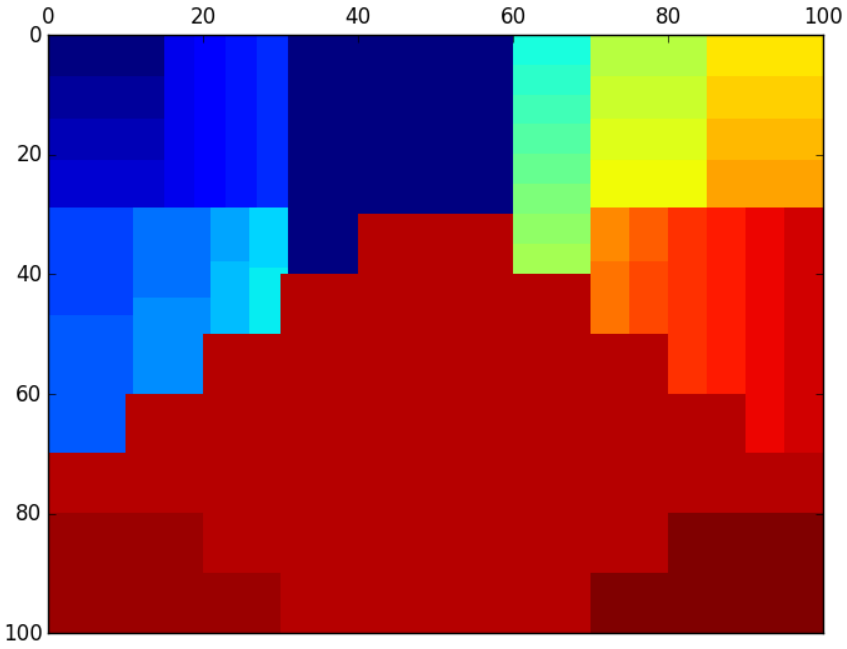

Figure A.8: Shape 9 after six splits

A.2 Random data

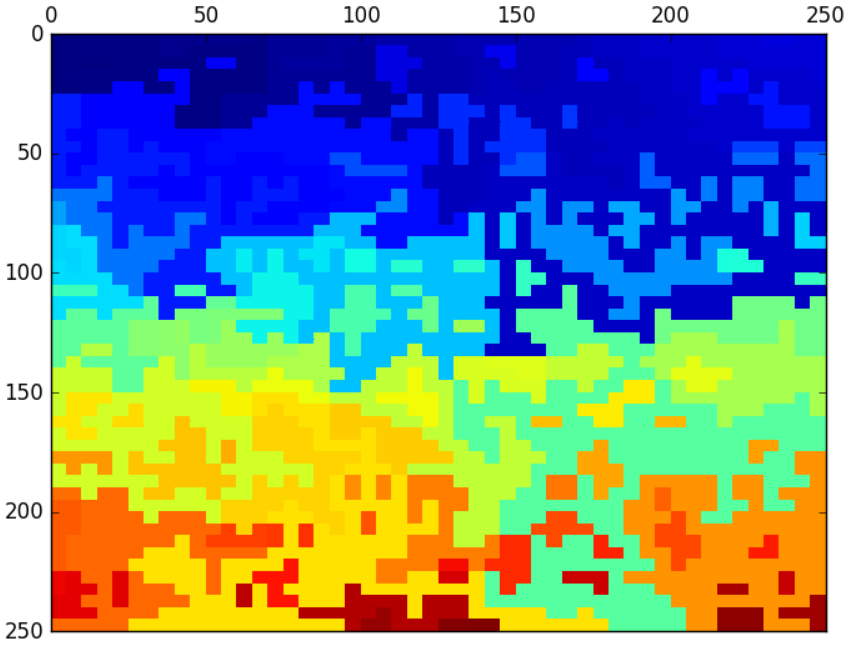

Figure A.9: 250x250 initial plot

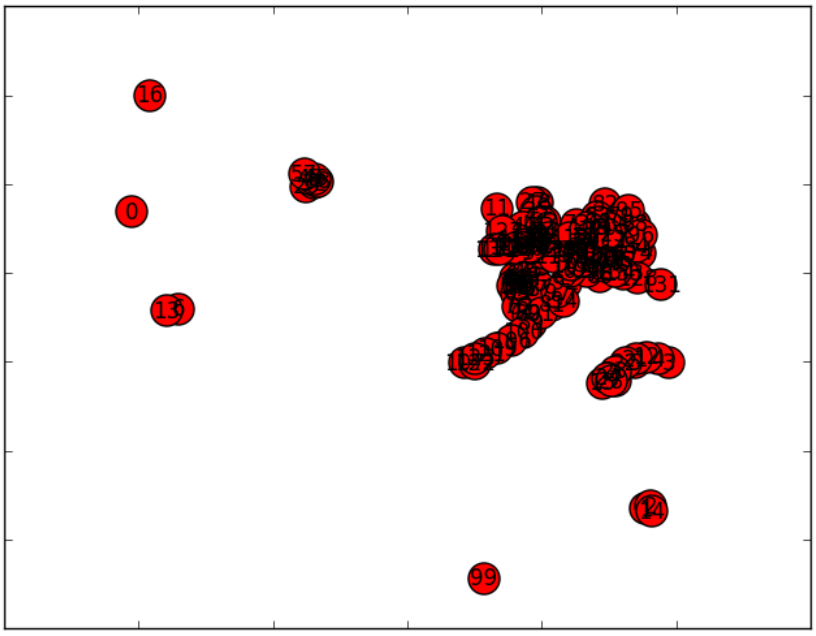

Figure A.10: 250x250 initial graph

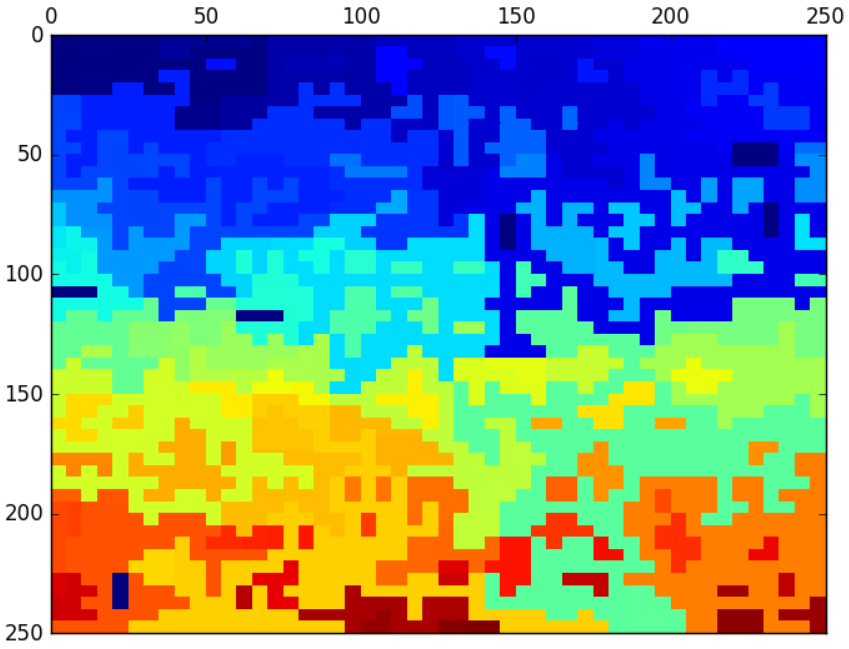

Figure A.11: 250x250 data after one split sweep

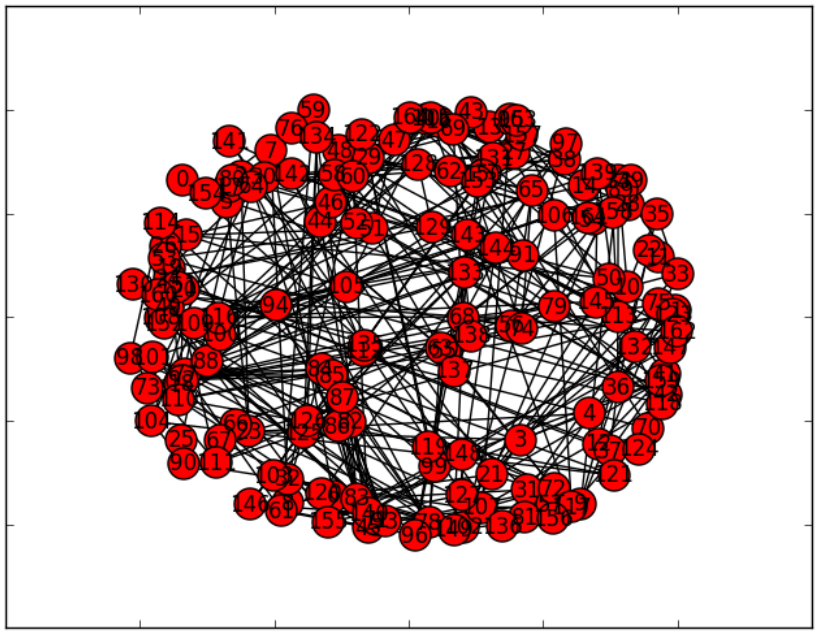

Figure A.12: 250x250 graph after one split

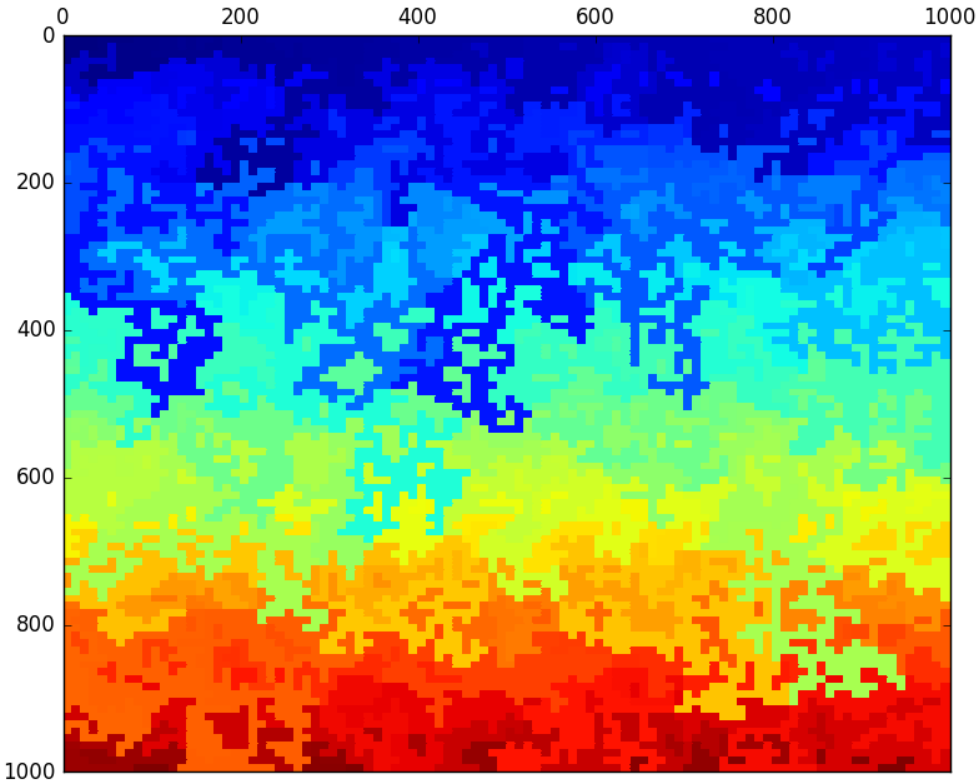

Figure A.13: 1000x1000 data initial plot

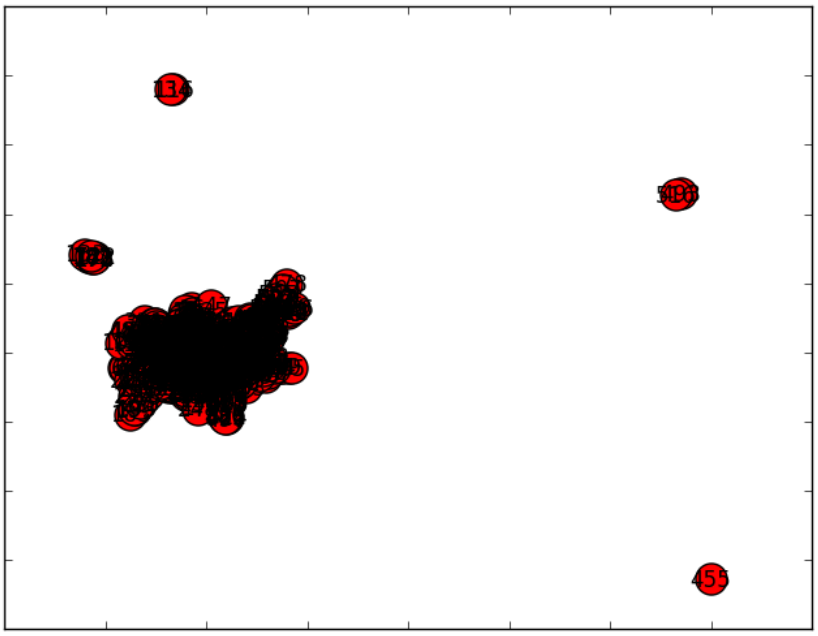

Figure A.14: 1000x1000 initial graph

A.3 Arkansas Data

Table of parameters:

Table A.1: Parameters for Arkansas data

Parameter	Value
NMI threshold	0.6
Distance threshold	50
β	4π
Number of splits	2
Minimum cluster size	40
Time steps	50
Number of simulations	10000

A.3.1 100x1000 data

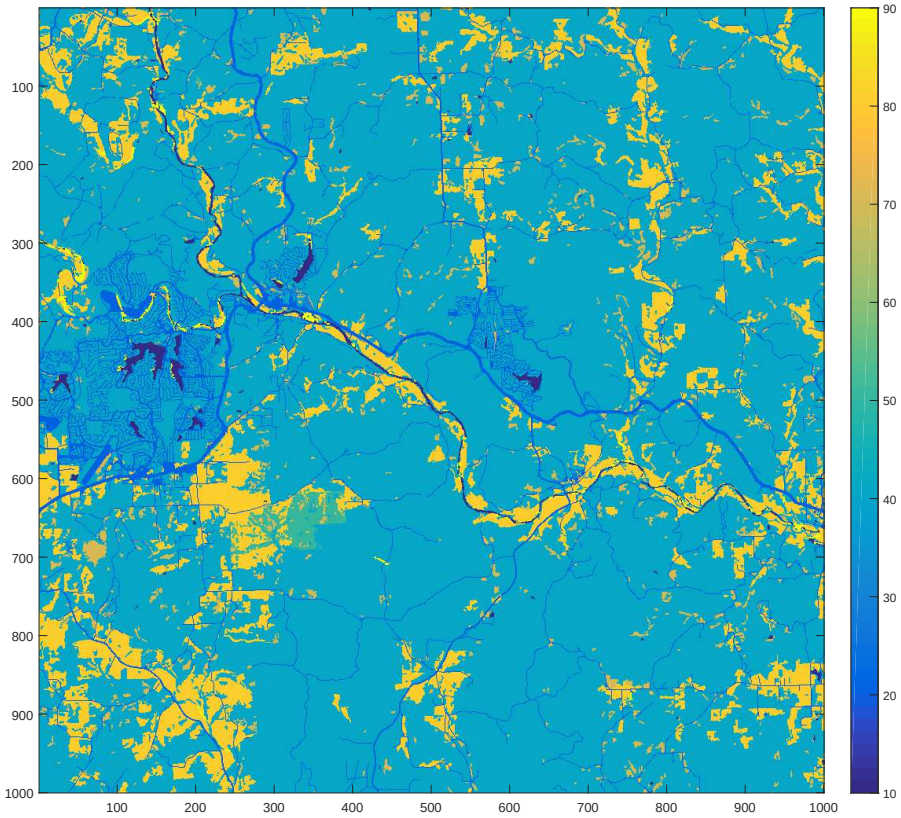

Figure A.15: 1000x1000 Arkansas Data raw plot

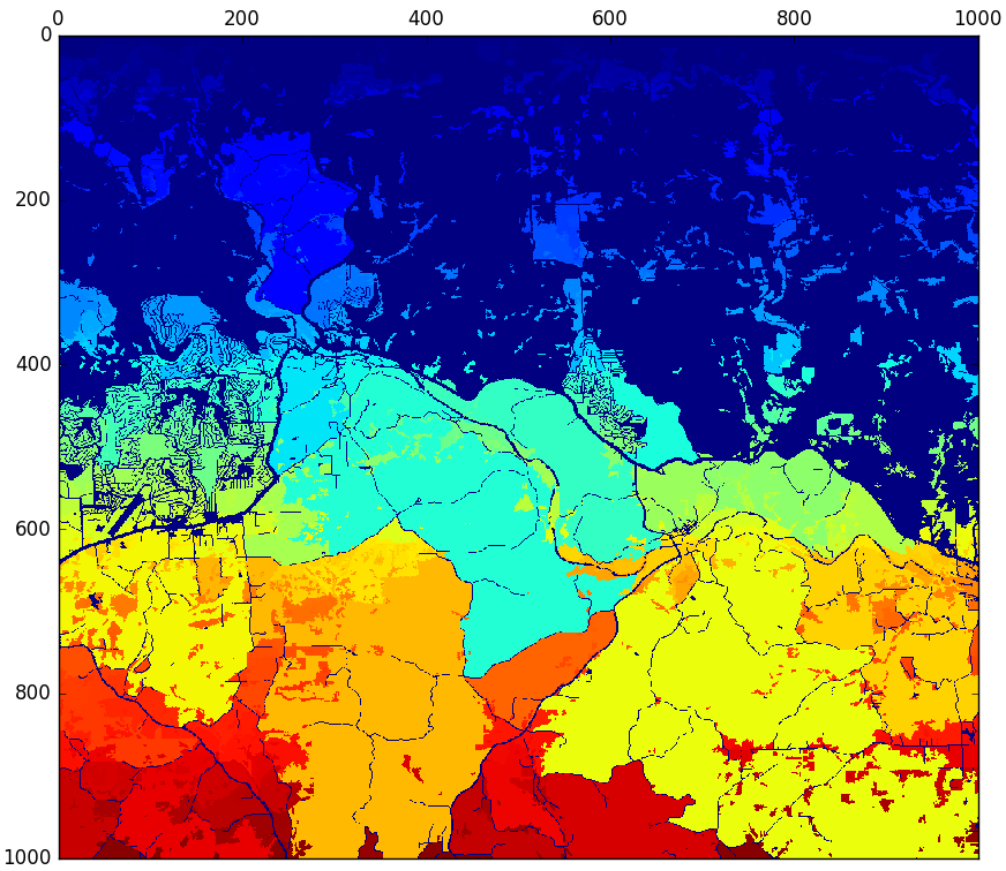

Figure A.16: 1000x1000 initial cluster

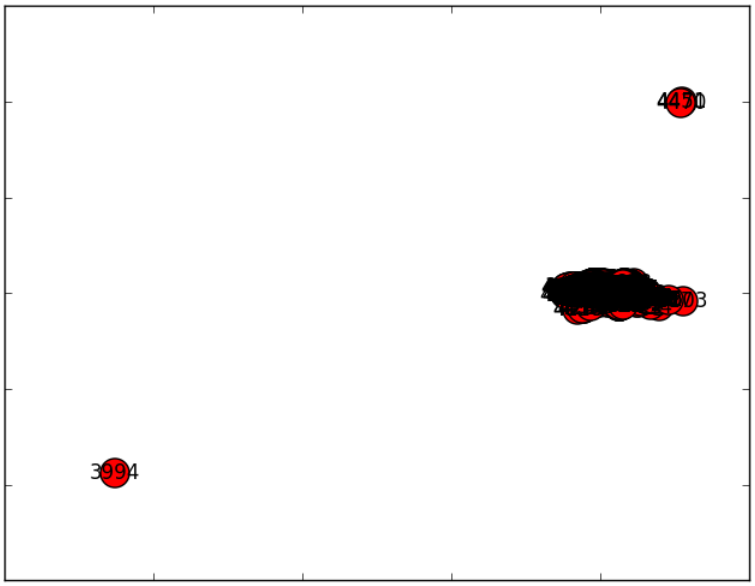

Figure A.17: 1000x1000 initial graph

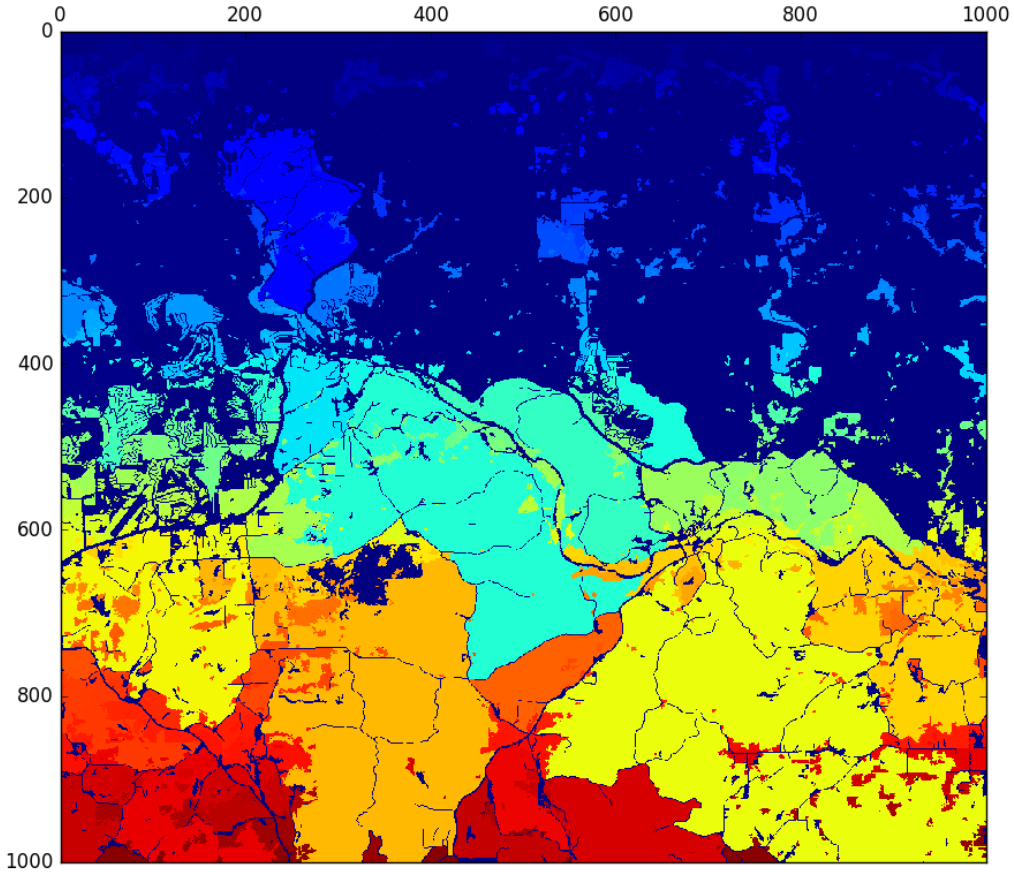

Figure A.18: 1000x1000 after one split

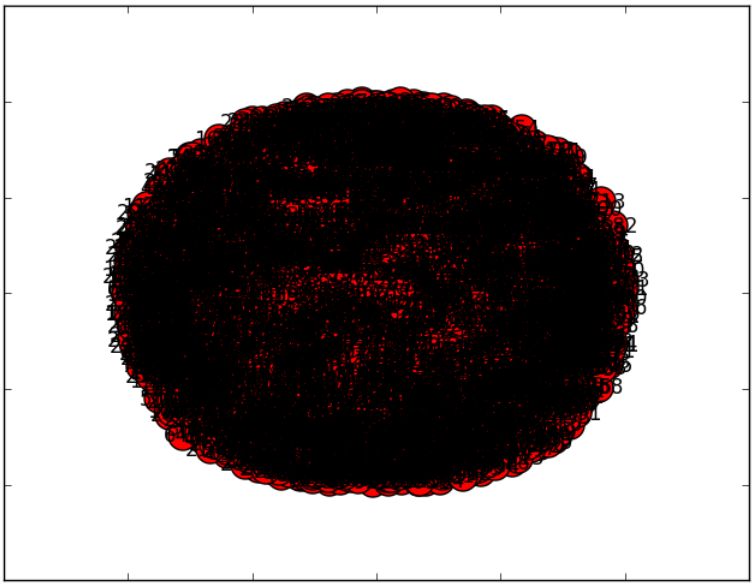

Figure A.19: 1000x1000 graph after one split

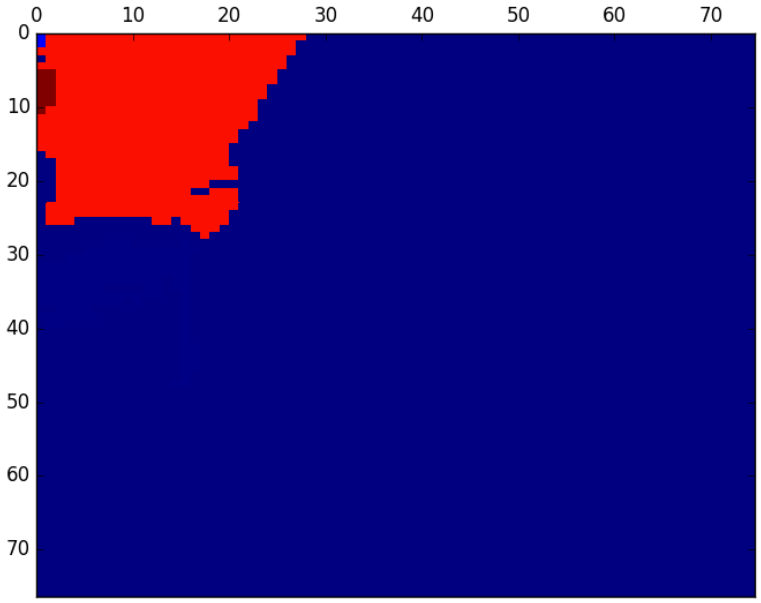

Figure A.20: 1000x1000 simulation results, zoomed

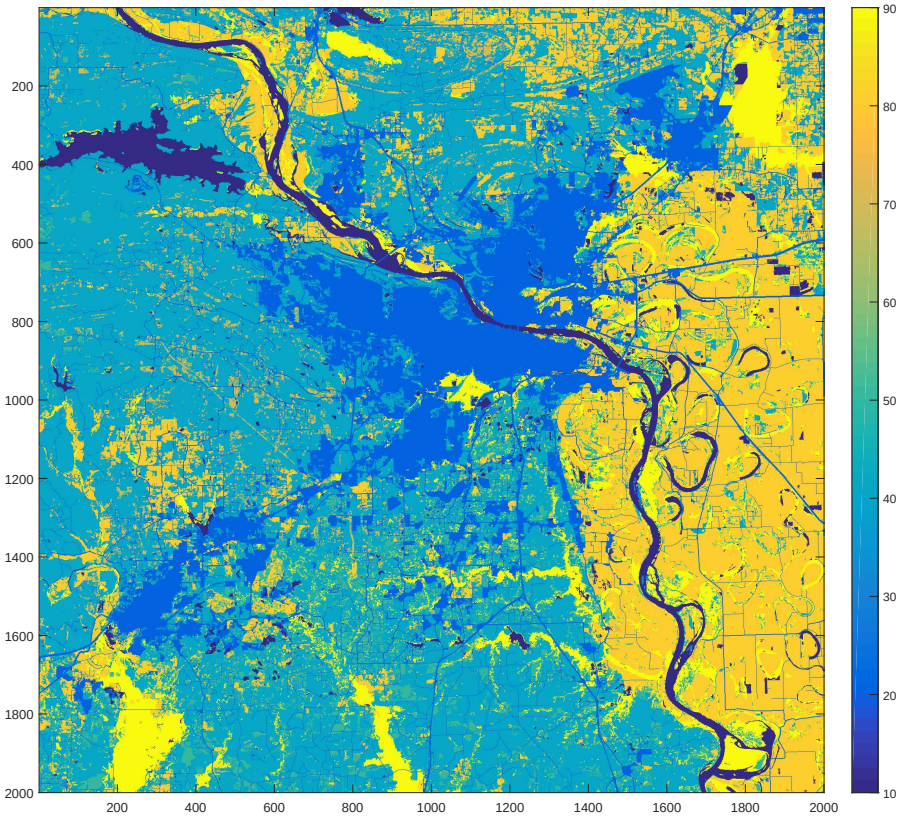

Figure A.21: 2000x2000 Arkansas Data

Appendix B

Summary and Explanation of Program

The package consists of two primary files, driver.py and cluster.py, that contain the core procedures of the method and a parameter file, cluster_parameters.py, that holds the necessary parameters to setup a run. The program was written in Python 3.5.1, utilizing the packages NumPy, NetworkX, and MatPlotLib. NumPy is the standard numerical mathematics package for Python, and the other two packages have excellent integration with NumPy. Source code for the package can be found on Bitbucket with the repository name dimension_reduction.

B.1 Driver.py

The driver.py file ties each routine specified in cluster.py together. It is worth noting that the clusters are stored using an OrderedDict container keyed on the unique cluster label; at the beginning simply the order in which they were found. The OrderedDict keeps the clusters in order of their label. After all splits are done, either by reaching the split limit or all clusters are above the NMI threshold, the clusters are relabeled with integers in successive order, zero indexed. This is to simplify the import into an adjacency matrix, using the clusters label as the index into the matrix. The driver file then creates the graph, and calls the Monte Carlo procedure, and displays the result.

B.2 Cluster_parameters.py

This is the parameters file that allows for easy changes to run parameters, as well as a couple function definitions that need to be easily changed. The parameter names and uses are in table B.1.

Table B.1: Parameter definitions

Parameter Name	Definition
NMI threshold	minimum NMI allowable
Distance threshold	maximum distance between centroids
Beta	parameter used in weight function
Number of splits	total number of splits allowable
Minimum cluster size	minimum cluster size allowable
Time steps	number of time steps to run in each simulation
Times to run	number of Monte Carlo simulations to run
Good values	list of good values to cluster

The 'Good values' field is a list of raw data values that we want to cluster over. This acts as a primitive filter to screen for unwanted data values, for instance the 'Developed' land classification.

Three function definitions are also included in this file (Table B.2):

Table B.2: Parameter definitions

Function name	Definition
norm	calculates norm between cluster centroids currently the Euclidean norm
wgt_function	weighting function applied to the distance adjacency matrix
prob_function	function to calculate movement probabilities from weighted matrix

B.3 Cluster.py

Cluster.py contains the majority of the procedures. A few critical procedures and definitions are detailed here, the rest of the details can be found in the source code comments.

B.3.1 Cluster

This is the class definition used to store each cluster and its relevant data, see Table B.3 for defintions.

Table B.3: Parameter definitions

Parameter	Usage
self.data_val	data value of the cluster that all cells share
self.label	cluster label
self.path	split path of cluster
self.cells	list of cells contained in cluster
self.num_cells	total number of cells in cluster
self.x_vals	average of cell x values, $\bar{x}$
self.y_vals	average of cell y values, $\bar{y}$
self.border	list of cells on border of cluster
self.border_size	size thereof
self.loc	location of cluster, as $(\bar{x}, \bar{y})$
self.start	cell where BFS started cluster
self.NMI	NMI value of cluster
self.adj_list	dict of clusters 'adjacent' to cluster
self.min_radius	minimum radius of cluster, see 2
self.times_visited	number of times visited during simulation
self.times_split	times cluster has been split
self.present	presence/absence data for simulation

B.3.2 Gen_BFS & BFS_cluster

These are the two breadth-first search routines. BFS_cluster finds the initial clusters off of contiguous data value alone. Gen_BFS is more complicated, as it also takes as an input a boundary list of cells that should split the cluster. That boundary is then assigned to one of the new clusters, and gen_BFS is then called on both sides of the boundary to find the two new clusters.

B.3.3 Cluster_split

This is the procedure that determines how and where to split the cluster. It has two sub functions, horz_sweep and vert_sweep. These conduct a sweep along the appropriate axes across the whole cluster that test each possible split, and find the maximum overall NMI

increase. Two new clusters are created with the appropriate cells, and the labels are set so that the split path is tracked. This is done in the manner shown in B.1. This method ensures a unique path, allowing each new cluster to be backtracked to its original cluster, and a unique label so there are not storage conflicts.

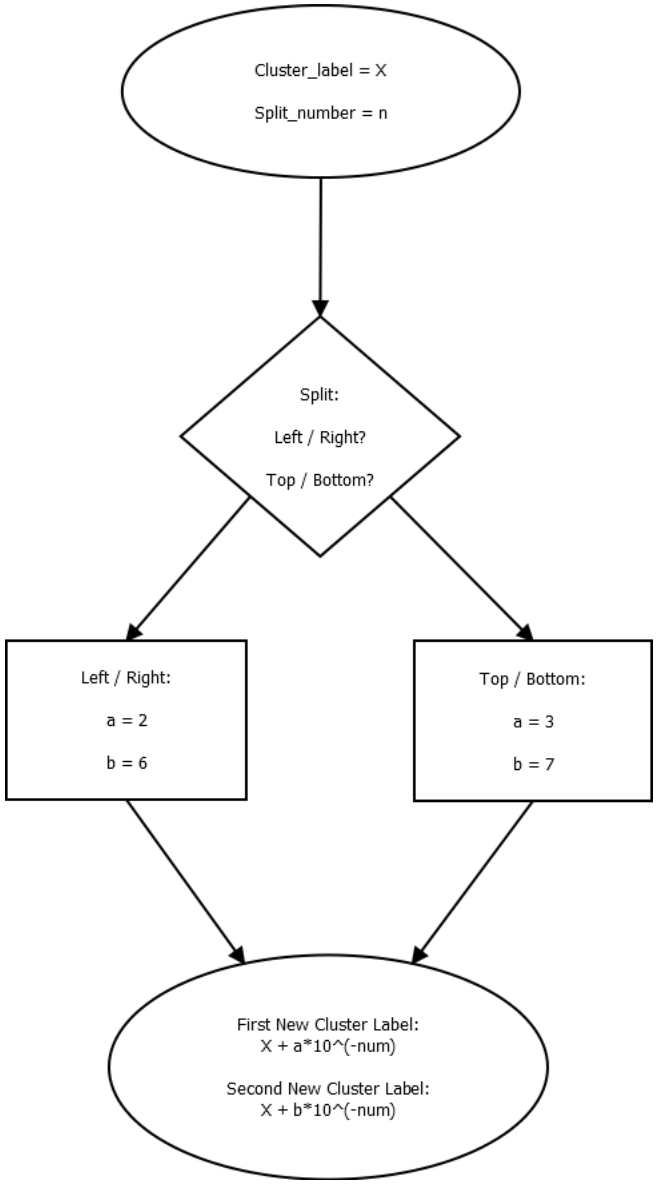

Figure B.1: Cluster labeling decision

Vita

Matthew Bachstein is a Masters Student at the University of Tennessee, Knoxville. He is a native of the Nashville area, and a graduate of John Paul II High School. He received his Bachelors of Science in Mathematical Sciences with General and Departmental honors from Clemson University in 2011. Matthews research interests currently consist of numerical mathematical methods with an eye towards incorporating graph theory, and a burgeoning interest in high performance and scientific computing.