5-20-2009

Mathematical methods for anomaly grouping in hyperspectral images

Timothy J. Doster

Mathematical Methods for Anomaly Grouping in Hyperspectral Images

by

Timothy J. Doster

B.S. Rochester Institute of Technology, 2009

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the School of Mathematical Sciences
Rochester Institute of Technology

May 20, 2009

Signature of the Author _____

Accepted by _____
Coordinator, M.S. Degree Program Date

SCHOOL OF MATHEMATICAL SCIENCES
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Timothy J. Doster
has been examined and approved by the
thesis committee as satisfactory for the
thesis required for the
M.S. degree in Applied and Computatational Mathematics

_____
Dr. David Ross, Thesis Advisor

_____
Dr. William Basener

_____
Dr. David Messinger

_____
Dr. Hossein Shahmohamad, Graduate Director

_____
Date

THESIS RELEASE PERMISSION
ROCHESTER INSTITUTE OF TECHNOLOGY
SCHOOL OF MATHEMATICAL SCIENCES

Title of Thesis:

Mathematical Methods for Anomaly Grouping in Hyperspectral Images

I, Timothy J. Doster, hereby grant permission to Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Signature _____

Date

Acknowledgments

First, I would like to thank my committee of Dr. David Ross, Dr. William Basener, and Dr. David Messinger. Dr. Ross has not only taught me the finer points of numerical analysis in the classroom, but, has found and guided me through this project (not to mention the grammar lessons he has taught me along the way). Dr. Basener, who has helped me to understand the details of the algorithms we explored by drawing countless diagrams on the white board. Dr. Messinger, who was always able to answer my questions about various imaging science concepts and who showed me that a mathematician can give a new understanding to an old problem.

I would also like to acknowledge some of the professors that I have had over the past five years who have shaped my academic development. Dr. Tamas Wiandt, for introducing me to my first research project and for always saying, "Any questions you have ... ask". Dr. DeLorenzo, for whom it was a pleasure to act as a teaching assistant. Dr. Anurag Agarwal, who took time out of his schedule to teach an independent study on algebraic number theory, and who allowed me to audit his cryptography class; both classes I found to have been exceptionally well taught. Professor David Bath-Hart, who taught the most demanding undergraduate courses with ease, and for always being there to explain a topic one, two, or three times.

My thesis committee and I would like to thank Dr. John Kerekes for supplying the Cooke City data (which can be found at http://dirsapps.cis.rit.edu/blindtest).

Lastly, I would again like to acknowledge Dr. Ross, Professor Barth-Hart, and Dr. Wiandt for writing me letters of recommendation for admission into graduate school, I would not have gotten into where I did if it were not for your words and advice.

To my father, mother, and loving fiancée Karen - without whom I could not have succeeded.

Mathematical Methods for Anomaly Grouping in Hyperspectral Images

by

Timothy J. Doster

Submitted to the
School of Mathematical Sciences
in partial fulfillment of the requirements
for the Master of Science Degree
at the Rochester Institute of Technology

Abstract

The topological anomaly detection (TAD) algorithm differs from other anomaly detection algorithms in that it does not rely on the data's being normally distributed. We have built on this advantage of TAD by extending the algorithm so that it gives a measure of the number of anomalous objects, rather than the number of anomalous pixels, in a hyperspectral image. We have done this by identifying and integrating clusters of anomalous pixels, which we accomplished with a graph-theoretical method that combines spatial and spectral information. By applying our method, the Anomaly Clustering algorithm, to hyperspectral images, we have found that our method integrates small clusters of anomalous pixels, such as those corresponding to rooftops, into single anomalies; this improves visualization and interpretation of objects. We have also performed a local linear embedding (LLE) analysis of the TAD results to illustrate its application as a means of grouping anomalies together. By performing the LLE algorithm on just the anomalies identified by the TAD algorithm, we drastically reduce the amount of computation needed for the computationally-heavy LLE algorithm. We also propose an application of a shifted QR algorithm to improve the speed of the LLE algorithm.

Contents

List of Figures

Chapter 1

Introduction

A hyperspectral image, in general, has hundreds of spectral bands in contrast to a multispectral image which has one to ten spectral bands. A normal digital image can be viewed as having three spectral bands (blue, red, and green), but in hyperspectral images a more complete part of the light spectrum is represented [11]. A regular digital image can be viewed as a collection of three-dimensional spectral vectors, each representing the information for one pixel. Similarly a hyperspectral image can be viewed as a collection of d-dimensional spectral vectors, each representing the information for one pixel. Hyperspectral images include spectral bands representing the visible, near infrared (0.7-1.0 micrometers), and short-wave infrared (1.0-3.0 micrometers). In Figure 1.1, a representation of the light spectrum is shown with the approximate coverage of a hyperspectral image.

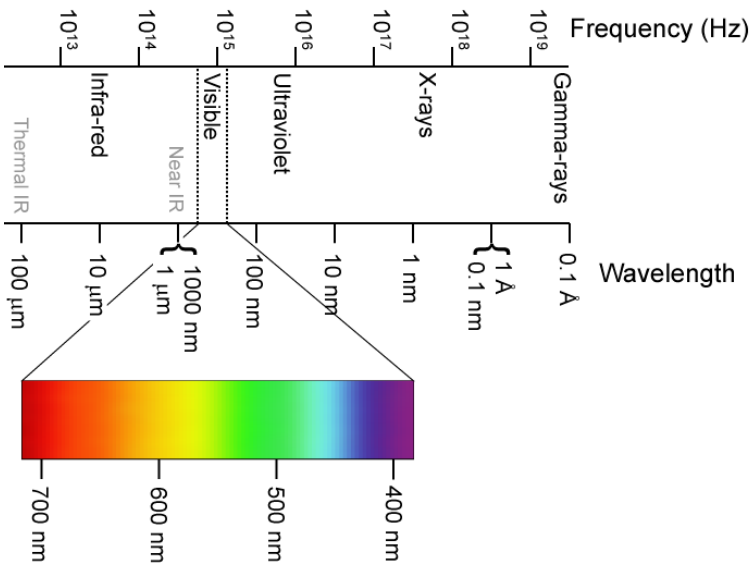


Figure 1.1: Electromagnetic Spectrum showing the visible, near-infrared, and shortwave infrared

Thus, hyperspectral images are favored over multispectral images for some applications such as forestry and crop analysis, as well as military exercises. The spectrum of vegetation, for example, is quite different from that of man-made objects even if painted to camouflage in with local vegetation. In This case, a simple photograph would not be able to pick out the man made objects as well as a hyperspectral image. A hyperspectral image can produce a traditional red-blue-green image by resampling the image using the human visual response.

Hyperspectral images are collected with special detectors that can be placed on high structures, flown in planes, or contained in satellites. The images used in this paper

were collected by a plane flying above the scene and reading hyperspectral data from the ground one line at a time for all the required bands. As the plane traveled, it recorded many lines and these were later assembled, with necessary smoothing done to remove effects from the uneven travel of the plane, into a complete hyperspectral image. The sensor aboard the plane worked by collecting the emitted solar radiation that is reflected off the ground or object on the ground. As the solar radiation enters the atmosphere, it is altered by the presence of water molecules and other particulate matter in the atmosphere as shown in Figure 1.2. The same effect happens once the solar radiation is reflected off the ground or object. The data that are recorded by the sensor are known as the radiance spectrum. The reflectance spectrum for a particular band is the ratio of the reflected radiation at that band to the incident radiation at that band, and can be recovered from the collected radiation spectrum by using atmospheric correction equations [7]. Throughout the paper we have chosen to use radiance images as they offer a uniform starting point for analyzing hyperspectral images since there is no one agreed-upon method for deriving reflectance spectra.

Clustering is the grouping of like pixels from an image based on their characteristics, typically their spectral response. The level of cluster differentiation is a choice of the user. For example, the user can choose to cluster all trees into one group or have a cluster of elms, pines, and oak trees. An anomalous pixel, for this research, is one that has some degree of dissimilarity from the rest of the pixels in the image. In more classic applications of anomaly detection Gaussian statistics are used - this however, from a theoretical aspect, requires that the image's pixels be normally distributed.

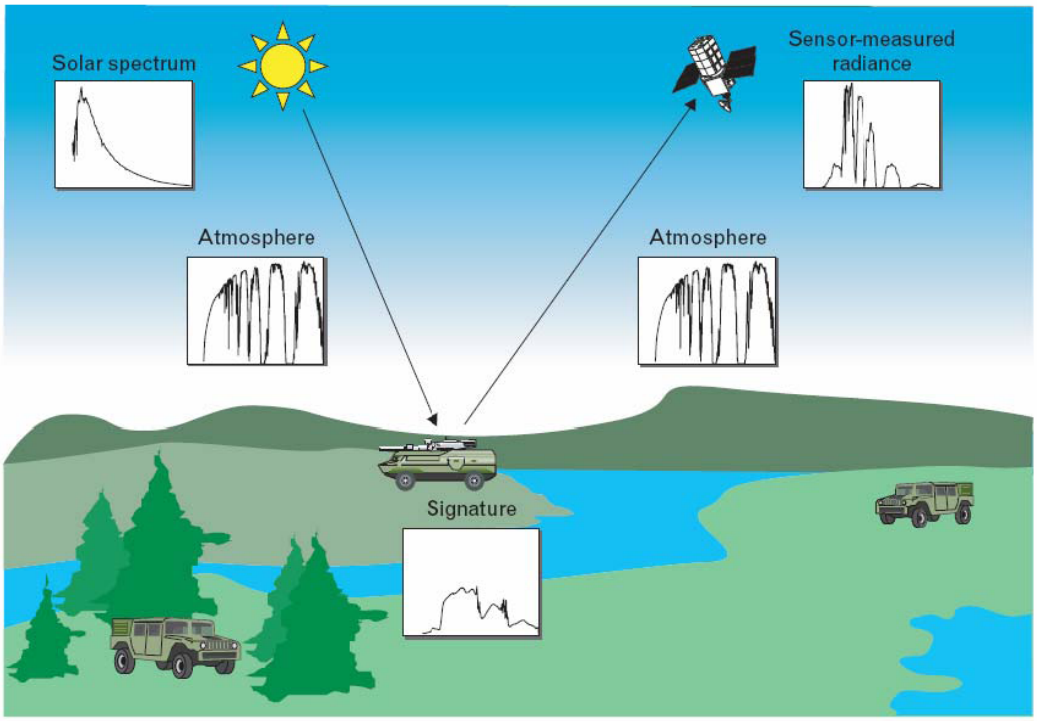


Figure 1.2: The path of solar radiation from the sun to the hyperspectral sensor (in this case on a satellite) [7]

For a naturally-occurring image, i.e., one that is not artificially created, this will not be the case as can be seen in Figure 1.3.

The most popular detection algorithms, which we will briefly describe in the next chapter, is the RX algorithm [7] which relies on Gaussian statistics. In Chapter 3, we will discuss the Topological Anomaly Detection (TAD) algorithm [2] which does not share this shortcoming. The output of the TAD algorithm, however, only declares anomalous pixels, it does not give a true count of the number of anomalous

objects in an image. For example, it may be advantageous to have all the anomalous pixels making up a camouflage net grouped and regarded as a single anomaly. The extension to the TAD algorithm discussed in Chapter 4 does this. It improves the visualization of anomalies by differentiating between point anomalies and those that belong to larger groups. In Chapter 5, we will discuss the use of local linear embedding to accomplish a similar goal. In Chapters 6 and 7, we will discuss the results and further work in this area.

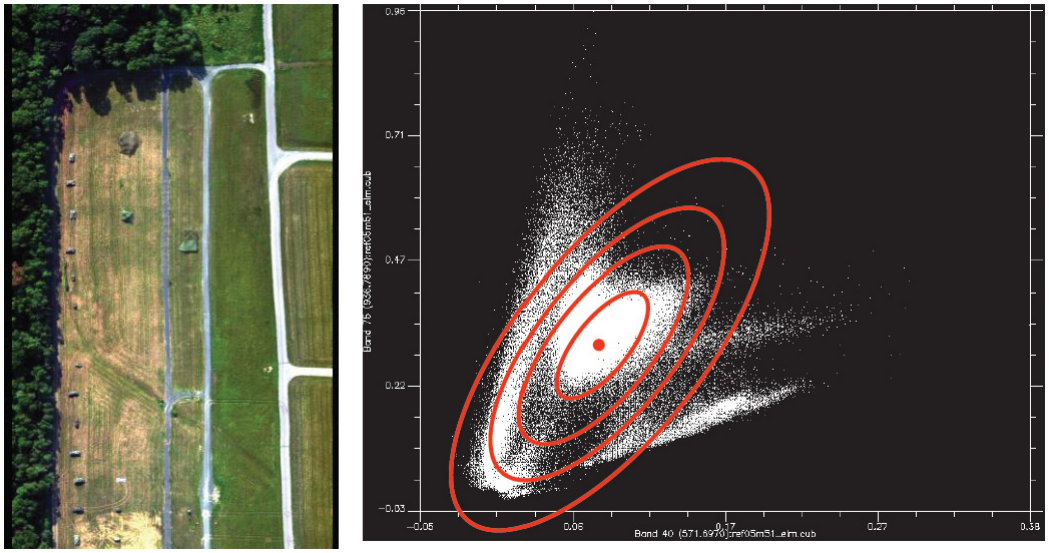


Figure 1.3: On the left a hyperspectral image shown with approximate red, blue, and green bands. On the right a scatter plot of the pixels from the image. Note how the data fails to fall into the ellipses centered at the mean [1].

1.1 Images

1.1.1 Cooke City

The image of Cooke City, Montana, in Figure 1.4, was collected in July of 2006 using a HyMap sensor operated by HyVista. It is an 800×280 pixel image that contains 126 spectral bands and has approximately 3 meter resolution[12]. In the image, there is a small town with several buildings, roads, cleared fields, and vehicles; the rest of the image is forest.

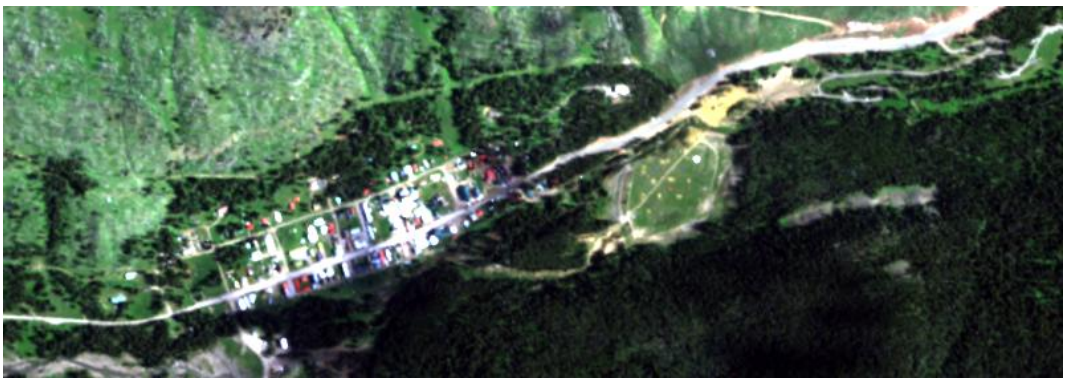


Figure 1.4: ENVI rendering of Cooke City image using approximate red, blue, and green bands

1.1.2 Copperas Cove

The image of Copperas Cove, Texas (sometimes referred to as URBAN in the literature), in Figure 1.5, was collected using a HYDICE sensor. It is a 307×307 pixel

image that contains 210 spectral bands, and has approximately 3 meter resolution. In the image we can see a large store with parking, a small housing division, and a large open field. For some of the analysis we used an 80×180 pixel subsection of the scene.

Figure 1.5: On the left ENVI rendering of Copperas Cove image using approximate red, blue, and green bands. On the right the image subset.

1.2 Software

We used ENVI 4.5, IDL 7.02, and MATLAB 2008B. We used ENVI 4.5 to display hyperspectral images and apply appropriate color mappings. We used IDL 7.02 to program the algorithms discussed in this thesis. To verify several of the outputs obtained in IDL 7.02, as well as for general computation, we used MATLAB 2008B.

1.3 Hardware

All of the calculations that we discuss in this thesis were done on a computer with a 2.0 GHZ Dual Core Intel processor with 1 GB of RAM running Windows XP.

Chapter 2

RX Algorithm

The RX algorithm developed by Reed and Yu [8] finds the mean of the data and identifies any pixels that have some greater than δ distance away from the mean as anomalies. Each pixel is surrounded by a sliding box that is centered on that pixel. The covariance of the data inside the box is then calculated. The rank of the pixel is the number of standard deviations by which that the pixel differs from the background; alternatively, the background model can be based on the entire image. The algorithm works well as long the image can be assumed to have normally-distributed data. Hyperspectral images, even after dimensionality reduction (mapping higher dimensionality data to lower dimensionality data but still preserving the most prominent features of the original high dimensionality data), frequently do not follow a Gaussian distribution. Another complication of the RX algorithm is determining the size of the sliding box without knowing the locations of the anomalies ahead of time.

2.1 Theorems and Definitions

Definition 2.1.1. *The expected value of a real valued discrete random variable X, $E(X)$, is the weighted sum of its expected outcomes or $\sum_i x_i p(x_i)$ where $p(x_i)$ is that $X = x_i$ [5].*

Definition 2.1.2. *The mean spectrum, $\mu(X)$, for a set of spectral vectors $X = \{X_1, X_2, \ldots, X_n\}$, where X_i is the spectrum of the i^{th} pixel, is $\mu(X) = \frac{1}{n}\sum_{i=1}^{n} X_i$.*

Definition 2.1.3. *Covariance is a measure of how much two real-valued random variables, X, Y, vary together, the covariance is defined as,*
$cov(X, Y) = E((X - E(X))(Y - E(Y)))$ [5].

Definition 2.1.4. *A covariance matrix is a matrix representing the pairwise covariances of a vector of real valued discrete random variables. For a hyperspectral image V, with a set of spectral vectors $X = \{X_1, X_2, \ldots, X_n\}$ and spectral mean μ the covariance matrix will be represented as $\Sigma = \frac{1}{N}\sum_{i=1}^{N} (X_i - \mu) \cdot (X_i - \mu)^T$.*

2.2 Algorithm

2.2.1 Step 0

Let a hyperspectral image, V, with N pixels and d spectral bands, be represented as a $N \times d$ matrix X whose i^{th} row, X_i, is the spectra of a pixel i.

2.2.2 Step 1

For pixel X_i we define $R(X_i)$ to be

$$R(X_i) = (X_i - \mu)^T \Sigma^{-1} (X_i - \mu),$$

where μ is the mean spectrum and Σ is the spectral covariance matrix. The inverse of the spectral covariance matrix is a change of basis into a new coordinate system where the variance of the data is one in every direction. When we calculate $R(X_i)$ we are finding the standard deviation in a multivariate sense.

2.2.3 Step 2

A threshold value, δ, is used to determine whether $R(X_i)$ is an anomaly or part of the background. If $R(X_i) < \delta$ then x is part of the background, if $R(X_i) \geq \delta$ then x is an anomaly. In the final output, pixels are assigned brightness values corresponding to the function defined in Step 1, the brighter the pixel the more anomalous it is.

2.3 Algorithm Results

In Figure 2.1, the RX algorithm (from ENVI) was run on the Cooke City and Copperas Cove images. It can be noted that in both sets of images that the algorithm

was able to find some of the anomalies but missed others. For example, in Cooke City some of the buildings are regarded as background while others are marked as anomalies, and in the cleared field not all the bare earth spots are marked as anomalous. In the Copperas Cove image, the RX algorithm does not perform well and marks several roof top pixels as anomalous.

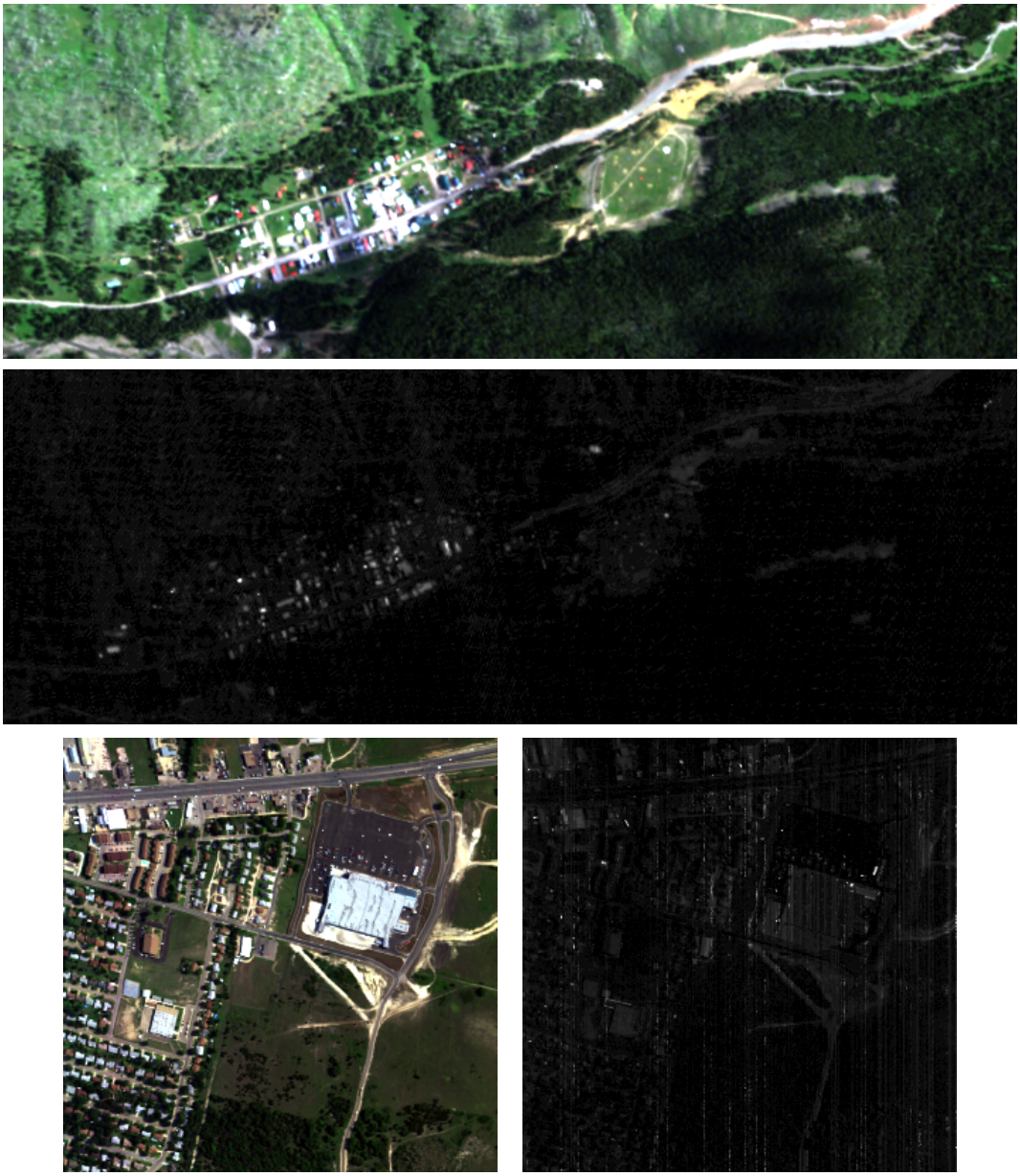


Figure 2.1: Top is the original Cooke City image. Middle is the RX algorithm rendering of the Cooke City image. Bottom left is the original Copperas Cove image, and bottom right is RX rendering of the Copperas Cove image.

Chapter 3

Topological Anomaly Detection Algorithm

The Topological Anomaly Detection algorithm [2] differs from the RX algorithm in that no assumption about the distribution of the data is made. It has been shown to perform better and more consistently than statistically-based anomaly detection algorithms such as the RX algorithm [3]. The algorithm has also been shown to perform particularly well as a method for characterizing the backgrounds of images. The TAD algorithm works by constructing a graph of the data and characterizing components of the graph, based on their sizes, as part of the background or as anomalies.

3.1 Theorems and Definitions

The following definitions are referenced from [4].

Definition 3.1.1. *A graph $G = (V, E)$ is a finite nonempty collection of objects, V, called vertices together with a set of pairs of distinct vertices, E, called edges.*

Definition 3.1.2. *A graph $H = (V', E')$ is a subgraph of a graph $G = (V, E)$, $H \subset G$, if $V' \subset V$ and $E' \subset E$.*

Definition 3.1.3. *For a graph $G = (V, E)$ with $u, v \in V$ a path between u and v is a set of unique vertices $w_1, w_2, \ldots, w_k \in V$ such that*
$(u, w_1), (w_1, w_2), \ldots, (w_{k-1}, w_k), (w_k, v) \in E$.

Definition 3.1.4. *A walk is an alternating sequence of vertices and edges; it removes the vertex (and edge) uniqueness that a path requires.*

Definition 3.1.5. *A graph $G = (V, E)$ is connected if for all $u, v \in V$ there exists a path between u and v.*

Definition 3.1.6. *A component of a graph, G, is a connected subgraph such that the vertex set and the edge set of the subgraph are proper subsets of the vertex set and edge set of graph, respectively.*

3.2 Algorithm

3.2.1 Step 0

Let X be a finite collection of k dimensional pixels.

3.2.2 Step 1

Construct the graph, G_r, where $r \in \mathbb{R}^+$ is some threshold. For G_r let X be the vertex set, and define the edge set, E, such that for u, $v \in X$, $uv \in E$ if and only if the spectral distance from pixel u to v is less than r.

3.2.3 Step 2

Let H be a component of G_r. Each component of G_r should represent a different type of material (for example grass, steel, trees) that is found in the image. If H contains at least $p\%$, defined as the background percentage, of the total pixels then it is part of the background of the image. We place the pixels contained in H into a set B. Typically, we let $p = 2$.

3.2.4 Step 3

We now calculate the rank of each pixel by summing the distances to its third, fourth, and fifth closest neighbors that are contained in B. This rank will be the measure of the anomalousness of each pixel. The final output will assign a scalar brightness to each pixel, the larger the scalar value the more anomalous the pixel is.

3.2.5 Run Time

The TAD algorithm can be run on a modern laptop, with a reasonable data set ($\sim 10^6$ pixels), in approximately two minutes.

3.3 Algorithm Results

In Figure 3.1, the TAD algorithm was run on the Cooke City and Copperas Cove images. It can be noted that in both sets of images, that the algorithm was able to find considerably more of the anomalies than the RX algorithm. For example, in Cooke City scene all buildings are marked as anomalies, as well as the bare earth spots and concrete circles outside of the city. In the Copperas Cove image, the TAD algorithm performs much better than the RX algorithm as it properly designated the buildings as anomalies, but marks the large shopping center as more anomalous. It also registers very few anomalous pixels on the cleared field.

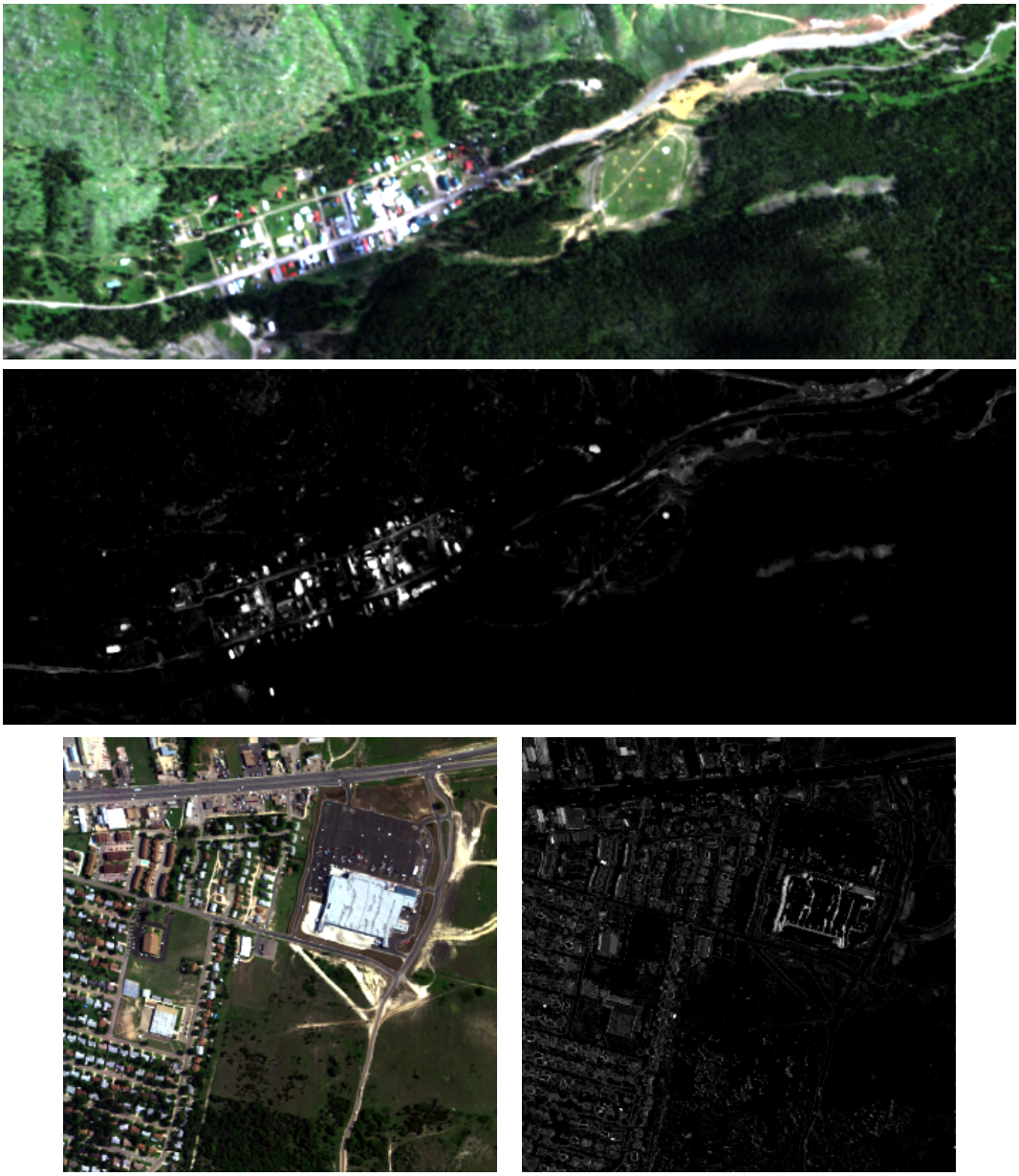


Figure 3.1: Top is original Cooke City image. Middle is TAD algorithm rendering of Cooke City image. Bottom left is original Copperas Cove image and bottom right is TAD algorithm rendering of Copperas Cove image.

3.4 Comparison of Performance in Relation to Other Algorithms

In Enhanced Detection and Visualization of Anomalies in Spectral Imagery [3], Basener showed by a range of examples that the TAD algorithm not only out performed the RX algorithm, but many of the variants of the RX algorithm (RXD, RXD local means, subspace RX, local RX, RXUTD, RXUTD local mean, RXD-UTD, and RXD-UTD local mean). In the comparison, five, hyperspectral images were used with more than 80 targets identified. Then by analyzing the findings using Receiver Operator Characteristic (ROC) curves (graphs that compare the probability of detection vs the probability of false alarm), Basener showed that TAD outperforms RX and its variants a majority of the time; some of these results can be seen in Figure 3.2. Based on these results, we have chosen to use TAD to produce the anomalous pixel sets used as a starting point for the algorithms discussed in the next two chapters.

	Probability of Detection				
Algorithm	**FR05m51**	**FR25m180**	**FR24m108**	**DR03m20**	**mean**
TAD	0.95	0.88	0.93	0.97	0.9325
RX	0.65	0.54	0.87	0.79	0.7125
Local RX	0.48	0.71	0.91	0.55	0.6625
SSRX	0.65	0.30	0.89	0.78	0.65

Figure 3.2: Shows the probability of detection at a 0.1 probability of false alarm in order from highest mean score to least mean score for 5 hyperspectral images [3].

Chapter 4

Anomaly Clustering Algorithm

We seek to improve the TAD algorithm by differentiating between point anomalies and anomalies consisting of several pixels. Challenges arise because of the uncertainty of the environment in which the image was taken - for example anomalies that we would like to group can have drastically different shapes (lines, polygons, etc.), and can be split between encompassing a whole pixel and part of a pixel. The anomaly clustering (AC) algorithm [6] extension works by post-processing the results of the TAD algorithm and building clusters of anomalous pixels that are both spatially contiguous and spectrally similar.

4.1 Theorems and Definitions

Definition 4.1.1. *The adjacency matrix [4] A for a graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_{|V|}\}$, is a $|V| \times |V|$, $0-1$ matrix that is populated such that*

$$A(i,j) = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}.$$

Theorem 4.1.2. *For a graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_k\}$, the number of distinct walks between v_i and v_j in t steps is equal to $A(i,j)^t$ where A is the adjacency matrix for G and $t \geq 1$.*

Proof. We shall denote this statement as $P(t)$ and proceed by using induction on t. For $t = 1$, $P(1)$ equates to $A^t = A$, which is the adjacency matrix and thus the number of walks between v_i and v_j is either 1 or 0 depending if they are connected which is the definition of $A(i,j)$.

Assume that $P(t)$ is true. By the inductive hypothesis $A^t(i,j)$ represents the number of distinct walks between v_i and v_j in t moves. We now calculate $A^{t+1} = (A^t)(A)$. By using a counting argument we can see that $A^{t+1}(i,j)$ gives the number of distinct walks between v_i and v_j in $t+1$ moves. Denote the i^{th} row of A^t by α and the j^{th} column of A by β. So $A^{t+1}(i,j) = \alpha \cdot \beta = \alpha_1 \times \beta_1 + \alpha_2 \times \beta_2 + \ldots \alpha_{|V|} \times \beta_{|V|}$. For any $1 \leq l \leq |V|$ we notice that α_l is the number of distinct walks between v_i and v_l in t steps and β_l is a $0, 1$ scalar denoting whether v_l and v_j are connected. So by computing $\alpha_l \times \beta_l$ we are computing the number of distinct walks of length $t+1$ that start with v_i and end with $v_l v_j$. $\alpha \cdot \beta$ gives the total number of distinct walks of length

$t+1$ between v_i and v_j in $t+1$ steps. Thus $P(t+1)$ is true, and by induction the result is shown. □

Corollary 4.1.3. *The total number of distinct walks between vertices v_i and v_j in at most r steps is $\sum_{l=1}^{r} A^l(i,j)$.*

Proof. By the content of Theorem 4.1.2 we saw that A^t gave the total number of distinct walks of length t; it is obvious that these walks cannot be repeated for any other t since they would not be of length t so we have $\sum_{l=1}^{r} A^l(i,j)$ as the number of distinct walks between vertices v_i and v_j in at most r steps. □

Corollary 4.1.4. *For a graph with a full set of self-loops ($(v_h, v_h) \in E \ \forall v_h \in V$)) there exists a walk between vertices v_i and v_j of length at most r, where $r = |V|$, if $A(i,j)^r \neq 0$.*

Proof. If there exists a walk of at most r length then this is trivial by the above Theorem 4.1.2. If there exists a walk of length $\rho > r$, then since there are only r vertices at least one the of the vertices is visited twice. Let this vertex be v_g, and represent the walk of length ρ as $v_i, v_{\rho,1}, \ldots, v_g, \ldots, v_\rho, \ldots, v_j$. This walk, though, can be shortened to $v_{\rho,i}, v_{\rho,1}, v_\rho, \ldots, v_j$ and it will still connect the same two endpoints. This can be repeated until we only have walks of length at most r. If there exists of walk of length $\alpha < r$ then this walk will be represented as a walk of length r with $r - \alpha$ self-loops of v_i at the start. □

4.2 Algorithm

4.2.1 Step 0

Let X be a finite collection of d-dimensional pixels. Let X' be the results of the TAD algorithm performed on X.

4.2.2 Step 1

For each pixel value $x \in X'$ we know that $x \in [0, 1]$. The larger the value of x, the more anomalous the pixel is. Let N be a subset of X containing only pixels whose corresponding value in X' are greater than some $\delta \in (0, 1)$. This allows us to pick out only the most anomalous pixels, and thus those that will be most interesting in the final analysis. For our work we typically used $\delta = 0.4$ as it gave good performance and allowed for quick runtime. Let k be the size of N.

4.2.3 Step 2

Let M be a $k \times k$ matrix, where each i^{th} row and column is associated with the i^{th} pixel contained in N, we will call it our detection matrix. Let i, $j \in N$. We define $M(i, j) = 1$ if i and j are connected. Here, we define connected as pixels that are spatially adjacent and within γ spectral radians (in spectral angle). Other metrics

for connectedness can also be developed, but we have not used any here. We define spatially adjacent strictly to mean pixels that share a common edge in the image. Otherwise $M(i, j) = 0$. Note that $M(i, i) = 1$.

4.2.4 Step 3

We now calculate M^t and reduce all nonzero entries to 1 and call this M_t or the t^{th} iteration matrix. For our purposes, we care only if there is a path between points i and j (not the number of paths). We iterate until we have reached solution equilibrium, that is until the $M_t = M_{t+1}$.

4.2.5 Step 4

In the i^{th} row of M_t, any non zero entries will belong to the same anomaly cluster as the i^{th} pixel of N since they can be reached in a finite number of moves from pixel i. This means that they are anomalous compared with the background but are connected, and estimated to be the same material based on the spectral angle measure. By identifying the pixels in the same anomaly group with a unique scalar, we color these groups accordingly. Unlike other detection algorithms, the scalar in the AC algorithm output, does not rank the pixels' anomalousness, it only serves to identity them as belonging to the same group. We will call the algorithm return the completed anomaly clustering map (though an uncompleted anomaly clustering map can be looked at for each iteration of Step 3).

For results see Chapter 6.

4.2.6 Runtime

The AC algorithm can be run on a modern laptop computer, for a reasonably-sized set of pixels ($\sim 10^6$ pixels), in approximately five minutes.

4.3 Example

The following simple example demonstrates how the AC algorithm works. For this example, let the following four-pixel-by-four-pixel grid (taken directly from an image preserving all pixel adjacencies) represent pixels we have determined to be anomalies and the values displayed be calculated spectral measures. Let $\gamma = 0.01$.

0.10	0.11	0.48	0.90
0.15	0.12	0.46	0.47
0.14	0.13	0.95	0.48
0.15	0.49	0.48	0.47

We now construct the detection matrix, designating the pixels in the grid from left to right, top to bottom:

$$M = M_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

which gives us the first iteration matrix. Since no two the rows of M_1 are identical, we cannot group any of the pixels together in this iteration, as seen in the following figure (first iteration anomaly clustering map).

0.10	0.11	0.48	0.90
0.15	0.12	0.46	0.47
0.14	0.13	0.95	0.48
0.15	0.49	0.48	0.47

We now calculate M^2:

$$M^2 = \begin{pmatrix} 2 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 4 & 2 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 2 & 0 & 0 & 2 & 3 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 1 & 2 & 3 \end{pmatrix},$$

and then by reducing all nonzero entries to 1 in M^2 we get the detection matrix for the second iteration:

$$M_2 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Since no two the rows of M_2 are identical we cannot group any of the pixels together again; as seen in the next figure (second iteration anomaly clustering map).

0.10	0.11	0.48	0.90
0.15	0.12	0.46	0.47
0.14	0.13	0.95	0.48
0.15	0.49	0.48	0.47

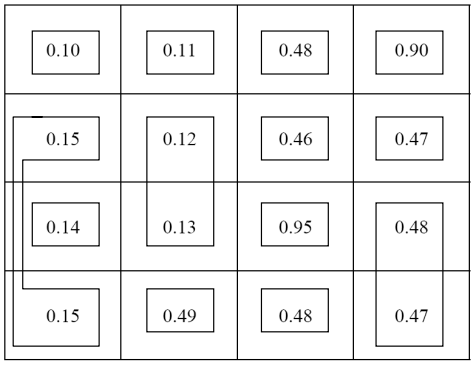
Seeing that $M_1 \neq M_2$, similarly we calculate M^3:

$$M^3 = \begin{pmatrix} 4 & 5 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 7 & 0 & 0 & 0 & 6 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 1 & 0 & 0 & 6 & 3 & 0 & 0 & 3 & 0 & 0 & 0 \\ 3 & 6 & 0 & 0 & 1 & 7 & 0 & 0 & 3 & 6 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 5 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 7 & 0 & 0 & 0 & 6 & 0 & 0 & 1 & 3 \\ 0 & 1 & 0 & 0 & 6 & 3 & 0 & 0 & 10 & 7 & 0 & 0 & 6 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 3 & 6 & 0 & 0 & 7 & 7 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 6 & 0 & 0 & 0 & 7 & 0 & 1 & 3 & 6 \\ 0 & 0 & 0 & 0 & 3 & 1 & 0 & 0 & 6 & 3 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 4 & 5 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 3 & 0 & 5 & 7 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 6 & 0 & 3 & 6 & 7 \end{pmatrix},$$

and then by reducing all nonzero entries to 1 we get the third iteration matrix, M_3,

$$M_3 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Using the third iteration matrix, we find third iteration anomaly clustering map:



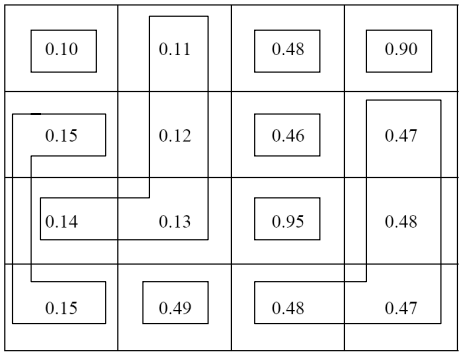
Note how we can now see the initial formation of anomaly groups. Since $M_3 \neq M_2$, we now calculate M^4:

$$M^4 = \begin{pmatrix} 16 & 20 & 0 & 0 & 0 & 13 & 0 & 0 & 1 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 20 & 29 & 0 & 0 & 1 & 25 & 0 & 0 & 5 & 14 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 17 & 5 & 0 & 0 & 25 & 14 & 0 & 0 & 13 & 0 & 0 & 0 \\ 13 & 25 & 0 & 0 & 5 & 30 & 0 & 0 & 15 & 25 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 16 & 20 & 0 & 0 & 0 & 13 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 20 & 29 & 0 & 0 & 0 & 25 & 0 & 1 & 5 & 14 \\ 1 & 5 & 0 & 0 & 25 & 15 & 0 & 0 & 44 & 30 & 0 & 0 & 25 & 0 & 0 & 0 \\ 5 & 14 & 0 & 0 & 14 & 25 & 0 & 0 & 30 & 31 & 0 & 0 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 13 & 25 & 0 & 0 & 0 & 30 & 0 & 5 & 14 & 25 \\ 0 & 1 & 0 & 0 & 13 & 5 & 0 & 0 & 25 & 14 & 0 & 0 & 17 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 5 & 0 & 16 & 20 & 13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 & 0 & 0 & 0 & 14 & 0 & 20 & 29 & 25 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 14 & 0 & 0 & 0 & 25 & 0 & 13 & 25 & 30 \end{pmatrix},$$

and then by reducing all nonzero terms to 1, we get the fourth iteration detection matrix, M_4:

$$M_4 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Using the fourth iteration matrix the fourth iteration anomaly clustering map is produced:



.

Again, note how the pixels are beginning to become grouped together. Comparing the fourth iteration anomaly map to the third iteration anomaly map we can see that

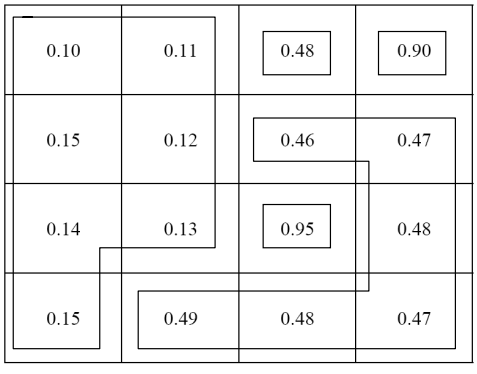
the larger groups are growing in size. Since again $M_4 \neq M_3$, we must calculate M^5:

$$M^5 = \begin{pmatrix} 21 & 30 & 0 & 0 & 1 & 25 & 0 & 0 & 5 & 14 & 0 & 0 & 1 & 0 & 0 & 0 \\ 30 & 46 & 0 & 0 & 5 & 44 & 0 & 0 & 16 & 30 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 0 & 0 & 26 & 15 & 0 & 0 & 45 & 30 & 0 & 0 & 25 & 0 & 0 & 0 \\ 25 & 44 & 0 & 0 & 15 & 51 & 0 & 0 & 35 & 46 & 0 & 0 & 15 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 21 & 30 & 0 & 0 & 0 & 25 & 0 & 1 & 5 & 14 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 & 46 & 0 & 0 & 0 & 44 & 0 & 5 & 15 & 30 \\ 5 & 16 & 0 & 0 & 45 & 35 & 0 & 0 & 81 & 60 & 0 & 0 & 45 & 0 & 0 & 0 \\ 14 & 30 & 0 & 0 & 30 & 46 & 0 & 0 & 60 & 56 & 0 & 0 & 30 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 25 & 44 & 0 & 0 & 0 & 51 & 0 & 14 & 30 & 45 \\ 1 & 5 & 0 & 0 & 25 & 15 & 0 & 0 & 45 & 30 & 0 & 0 & 26 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 & 0 & 0 & 0 & 14 & 0 & 21 & 30 & 25 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 15 & 0 & 0 & 0 & 30 & 0 & 30 & 46 & 44 \\ 0 & 0 & 0 & 0 & 0 & 0 & 14 & 30 & 0 & 0 & 0 & 45 & 0 & 25 & 44 & 51 \end{pmatrix},$$

and then by reducing all nonzero terms to 1, we get the fifth iteration detection matrix, M_5:

$$M_5 = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Using the fifth iteration matrix, the fifth iteration anomaly clustering map is produced:



.

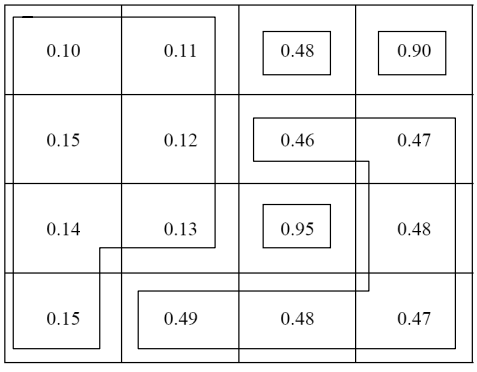
Since $M_5 \neq M_4$ we continue and calculate M^6:

$$M^6 = \begin{pmatrix} 51 & 76 & 0 & 0 & 6 & 69 & 0 & 0 & 21 & 44 & 0 & 0 & 6 & 0 & 0 & 0 \\ 76 & 120 & 0 & 0 & 21 & 120 & 0 & 0 & 56 & 90 & 0 & 0 & 21 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 21 & 0 & 0 & 71 & 50 & 0 & 0 & 126 & 90 & 0 & 0 & 70 & 0 & 0 & 0 \\ 69 & 120 & 0 & 0 & 50 & 141 & 0 & 0 & 111 & 132 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 51 & 76 & 0 & 0 & 0 & 69 & 0 & 6 & 20 & 44 \\ 0 & 0 & 0 & 0 & 0 & 0 & 76 & 120 & 0 & 0 & 0 & 120 & 0 & 20 & 50 & 89 \\ 21 & 56 & 0 & 0 & 126 & 111 & 0 & 0 & 231 & 176 & 0 & 0 & 126 & 0 & 0 & 0 \\ 44 & 90 & 0 & 0 & 90 & 132 & 0 & 0 & 176 & 162 & 0 & 0 & 90 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 69 & 120 & 0 & 0 & 0 & 140 & 0 & 44 & 89 & 126 \\ 6 & 21 & 0 & 0 & 70 & 50 & 0 & 0 & 126 & 90 & 0 & 0 & 71 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 20 & 0 & 0 & 0 & 44 & 0 & 51 & 76 & 69 \\ 0 & 0 & 0 & 0 & 0 & 0 & 20 & 50 & 0 & 0 & 0 & 89 & 0 & 76 & 120 & 120 \\ 0 & 0 & 0 & 0 & 0 & 0 & 44 & 89 & 0 & 0 & 0 & 126 & 0 & 69 & 120 & 140 \end{pmatrix},$$

and then by reducing all nonzero terms to 1, we get the sixth iteration detection matrix, M_6:

$$M_6 = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Using the sixth iteration matrix, the sixth iteration anomaly clustering map is produced:



Since $M_6 \neq M_5$ we continue and calculate M^7:

$$M^7 = \begin{pmatrix}
127 & 196 & 0 & 0 & 27 & 189 & 0 & 0 & 77 & 134 & 0 & 0 & 27 & 0 & 0 & 0 \\
196 & 316 & 0 & 0 & 77 & 330 & 0 & 0 & 188 & 266 & 0 & 0 & 77 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
27 & 77 & 0 & 0 & 197 & 161 & 0 & 0 & 357 & 266 & 0 & 0 & 196 & 0 & 0 & 0 \\
189 & 330 & 0 & 0 & 161 & 393 & 0 & 0 & 343 & 384 & 0 & 0 & 161 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 127 & 196 & 0 & 0 & 0 & 189 & 0 & 26 & 70 & 133 \\
0 & 0 & 0 & 0 & 0 & 0 & 196 & 316 & 0 & 0 & 0 & 329 & 0 & 70 & 159 & 259 \\
77 & 188 & 0 & 0 & 357 & 343 & 0 & 0 & 659 & 518 & 0 & 0 & 357 & 0 & 0 & 0 \\
134 & 266 & 0 & 0 & 266 & 384 & 0 & 0 & 518 & 470 & 0 & 0 & 266 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 189 & 329 & 0 & 0 & 0 & 386 & 0 & 133 & 259 & 355 \\
27 & 77 & 0 & 0 & 196 & 161 & 0 & 0 & 357 & 266 & 0 & 0 & 197 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 26 & 70 & 0 & 0 & 0 & 133 & 0 & 127 & 196 & 189 \\
0 & 0 & 0 & 0 & 0 & 0 & 70 & 159 & 0 & 0 & 0 & 259 & 0 & 196 & 316 & 329 \\
0 & 0 & 0 & 0 & 0 & 0 & 133 & 259 & 0 & 0 & 0 & 355 & 0 & 189 & 329 & 386
\end{pmatrix},$$

and then by reducing all nonzero terms to 1, we get the seventh iteration detection matrix, M_7:

$$M_7 = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

We now see that $M_7 = M_6$ so we no longer iterate. The anomaly clustering map found in iteration six is the final iteration anomaly map, and the algorithm will return it with appropriate scalar markings of the anomaly clusters. The algorithm was able to show that the pixel grid has two large anomalies (likely composed of the same material), one with six pixels and the other with seven, and it has three single-pixel anomalies as well. This was far more information then was known before running the algorithm, so we can judge from this example that the AC algorithm was a success and worth the run time.

Chapter 5

Local Linear Embedding

Local Linear Embedding (LLE) [10, 9] is a nonlinear manifold-based approach to clustering. This differs from the previously mentioned RX algorithm which relies upon linear boundaries between clustered regions of pixels. Real-world hyperspectral data does not form linear regions of pixels, so a method that does not rely on this assumption offers many advantages. LLE relies on the fact that if there is enough data so that each point in the data set will lie with its neighbors close to a locally linear patch of the manifold. A drawback to LLE is the requirement of calculating the inverse of the covariance matrix, for each pixel as well as the full spectrum of a matrix that has dimensions of the total number of pixels in the image. Both of these make LLE computation costly. By performing LLE only on the anomalous pixels detected by the TAD algorithm, one can reduce the computation involved drastically as compared to performing LLE on the whole image. We also offer a method to

reduce the amount of computation that goes into calculating the spectrum.

5.1 Theorems and Definitions

Definition 5.1.1. *A manifold is a topological space that is locally Euclidean, in other words for every point there is a neighborhood that is homeomorphic to an open sphere in* $\mathbb{R}^N$.

Definition 5.1.2. *The QR factorization of a matrix* $A \in \mathbb{R}^{n\times n}$ *is* $A = QR$ *where* Q *is an* $n \times n$ *matrix of orthonormal columns and* R *is a* $n \times n$ *upper triangular matrix which are formed by:*

$$\begin{aligned} a_1 &= r_{11}q_1 \\ a_2 &= r_{12}q_1 + r_{22}q_2 \\ &\vdots \\ a_n &= r_{1n}q_1 + r_{2n}q_2 + \cdots + r_{nn}q_n \end{aligned}$$

where a_i, q_i *is the* i^{th} *column of* A *and* Q *respectively and* $r_{jk} = R(j,k)$ *[13].*

Definition 5.1.3. *The QR algorithm applied to a matrix A is as follows:*

$$A^{(0)} = A.$$

$$\text{For } k = 1, 2, \ldots$$

$$Q^{(k)}R^{(k)} = A^{(k-1)}$$

$$A^{(k)} = R^{(k)}Q^{(k)},$$

where $B^{(t)}$ is the t^{th} iteration of matrix B [13].

5.2 Algorithm

5.2.1 Step 0

Let $X = \{X_1, X_2, \ldots, X_N\}$ be a set of vectors with $X_i \in \mathbb{R}^n$.

5.2.2 Step 1

Calculate the k nearest neighbors for each $x \in X$ using the Euclidean metric (though other metrics could be used). We will denote these nearest neighbors in a $n \times k+1$ matrix M such that $M(i,1) = X_i$ and $M(i,2), M(i,3), \ldots, M(i,k+1)$ are the k nearest neighbors of X_i.

5.2.3 Step 2

We now calculate the reconstruction weights for each X_i. By using the cost function:

$$E(W) = \sum_{i=1}^{N} |X_i - \sum_{j \neq i} W_{i,j} X_j|^2$$

we can calculate the squared difference between each point and its neighbors. To find $W(i, j)$, the cost function is minimized subject to $W(i, k) = 0$ if X_k is not a neighbor of X_i and $\sum_{k=1}^{N} W(i, k) = 1$. By forcing the weights to sum to 1 we are removing the effects of translations of points. The use of the cost function ensures that points are not dependent upon rotations and rescaling. Now the set of weights will represent the underlying geometric properties of the data set.

5.2.4 Step 3

Now by use of a similar cost function we will map each X_i to a lower dimensional Y_i. The cost function mentioned is:

$$\Phi(Y) = \sum_{i=1}^{N} |Y_i - \sum_{j \neq i} W_{i,j} Y_j|^2,$$

and we minimize it by fixing $W(i, j)$ and optimizing Y_j. We now find the $L + 1$ smallest eigenvalues, $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{L+1}$, and their corresponding eigenvectors, $V_1, V_2, \ldots, V_{L+1}$. We reject the smallest eigenvector as it is the unit vector with eigenvalue 0. Now we use the remaining eigenvectors to project X from n dimensions to

L dimensions: $X_i \mapsto (V_2(i), V_3(i), \ldots, V_{L+1}(i))$.

For results see Chapter 6.

5.2.5 Modifications

For our purposes we will start with a Step 0 where a hyperspectral image V with a set of spectral vectors Z. We will let Z' be the results of the TAD algorithm performed on Z. Now let X be the spectral vectors in Z that have corresponding values in Z' greater than some threshold $\delta \in (0, 1)$. We will now proceed to Step 1 and use the set X which we have defined.

5.3 Improved Calculation of Embedding Eigenvectors

Utilizing LLE with hyperspectral images we would like to project down from N dimensions to three: one each for red, blue, and green. Since only the four smallest eigenvalues and corresponding eigenvectors are required in this case, we offer an application of the QR algorithm [13] to quickly calculate these eigenvalues and eigenvectors. By only calculating these eigenvalues instead of the complete spectrum we save dramatically on computation time.

5.3.1 Step 1

The matrix for which we want to find the four smallest eigenvalues and corresponding eigenvectors we will denote M. Run the QR algorithm on M until $|R^{(k)}(1,1) - R^{(k+1)}(1,1)| < \delta$ for a suitably small $\delta \in \mathbb{R}^+$. Let $\omega = R^{k+1}(1,1)$. Since the QR algorithm can be shown to produce the same output step for step as the power method and the power method finds the largest eigenvalue we know ω is the largest eigenvalue in the spectrum. We will use ω to shift the spectrum so the smallest eigenvalues are now the greatest eigenvalues. It should be noted that if δ is not chosen to be small enough then the estimate of ω will not be accurate enough to produce the correct shift.

5.3.2 Step 2

Let $M' = M - \omega I$, where I is the identity matrix. Now perform the QR algorithm on M' until $|R'^{(l)}(4,4) - R'^{(l+1)}(4,4)| < \varepsilon$ for a suitably small $\varepsilon \in \mathbb{R}^+$. We expect $\delta > \varepsilon$.

5.3.3 Step 3

Now we remove the shift and let $M'' = M' + \omega I$. Let λ_i and V_i be our desired eigenvalue and corresponding eigenvector pairs for $i = 1, 2, 3, 4$. Then,

$$\begin{aligned} \lambda_1 &= R'(1,1) + \omega \quad V_1 = Q_1'^{(l+1)} \\ \lambda_2 &= R'(2,2) + \omega \quad V_2 = Q_2'^{(l+1)} \\ \lambda_3 &= R'(3,3) + \omega \quad V_3 = Q_3'^{(l+1)} \\ \lambda_4 &= R'(4,4) + \omega \quad V_4 = Q_4'^{(l+1)} \end{aligned},$$

where Q_i' is the i^{th} column of Q.

Chapter 6

Results

In Figure 6.1 we see the original image, the TAD output, and the AC output for Cook City. As discussed previously the TAD algorithm did a far better job then the RX algorithm in identifying all the roof tops as anomalies, as well as the cleared fields and the cement circles. The AC algorithm builds on this by showing the buildings as distinct anomalous objects instead of all looking similar. The AC algorithm groups the anomalous pixels that make up the cleared field in the top middle of the image correctly into one anomalous object (through a combination of Google Earth and spectral properties of the pixels we determined that this should be considered one object). It should be noted that there are several holes in the roof tops of the AC algorithm output, and this is a result of the thresholding that was needed to allow the algorithm to run in the desired time. For the image, the TAD algorithm identified 4951, anomalies but with the AC algorithm extension we learn that there are 67

anomalous objects in the image.

In Figure 6.2, we again see the original image, the TAD output, and the AC output for Copperas Cove. Similar to the Cook City image the TAD algorithm did a better job than the RX algorithm in identifying the anomalies in the image. In this image, we see the improvement of the AC algorithm more clearly. In the second column of Figure 6.2, we see a small subsection of the image. It contains, which can be seen partially in the RGB image, five houses along a road. The TAD algorithm correctly identifies these houses as anomalies but it is difficult to discern if they are similar or distinct to one another. The AC algorithm, however, clearly shows that these are five distinct anomalous objects (houses), and even shows that they have driveways that are distinct from the houses. For the image, the TAD algorithm identified 1271 anomalies but with the AC algorithm extension we learned that there are 44 anomalous objects in the image.

In Figure 6.3 we see the Cooke City image at the top and the TAD rendering in the middle. Thresholding the results of the TAD rendering and running LLE on these we get results seen in the bottom of the figure. These results prove interesting because LLE has classified most of the town and the cleared fields as blueish yellow except for the four boxed areas where they are pink. In the Figure 6.4, we explore these four subsections of the image and look at the spectral properties of the pixels that LLE designated pink. Looking at the spectrum graphs in column three of Figure 6.4, we

see that all four subsections have nearly identical spectral properties. Investigating this further with the use of Google Earth we are able to determine that these houses all have painted tin roofs. By running LLE only on the anomalies obtained from the TAD algorithm, we are able to learn that these four anomalous objects were all composed of the same material, and are vastly different in a spectral sense than the rest of the scene.

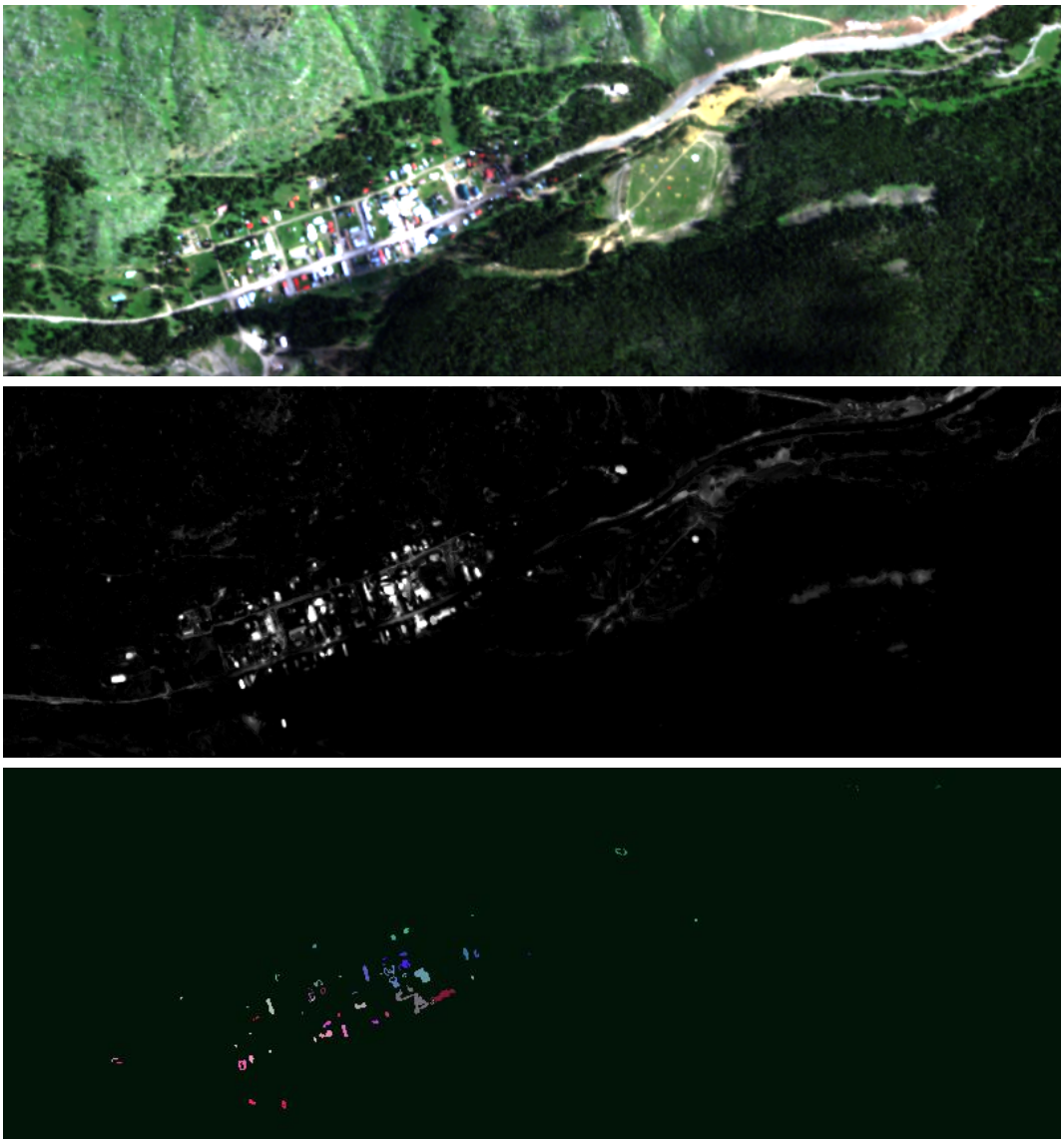


Figure 6.1: From top to bottom the original image, the TAD rendering of the image, and the anomaly clustering algorithm of the original image

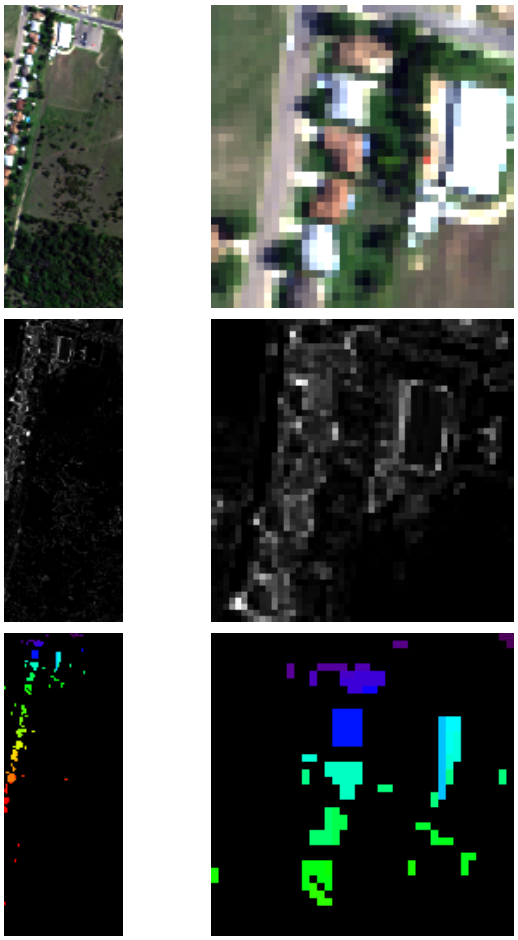


Figure 6.2: Top is the subset of the Copperas Cove image and a further subset of the top right of the image. Middle and bottom is the TAD and AC algorithm rendering of the Copperas Cove subset image and with the same zoomed subset of the top right part of the image, respectively.

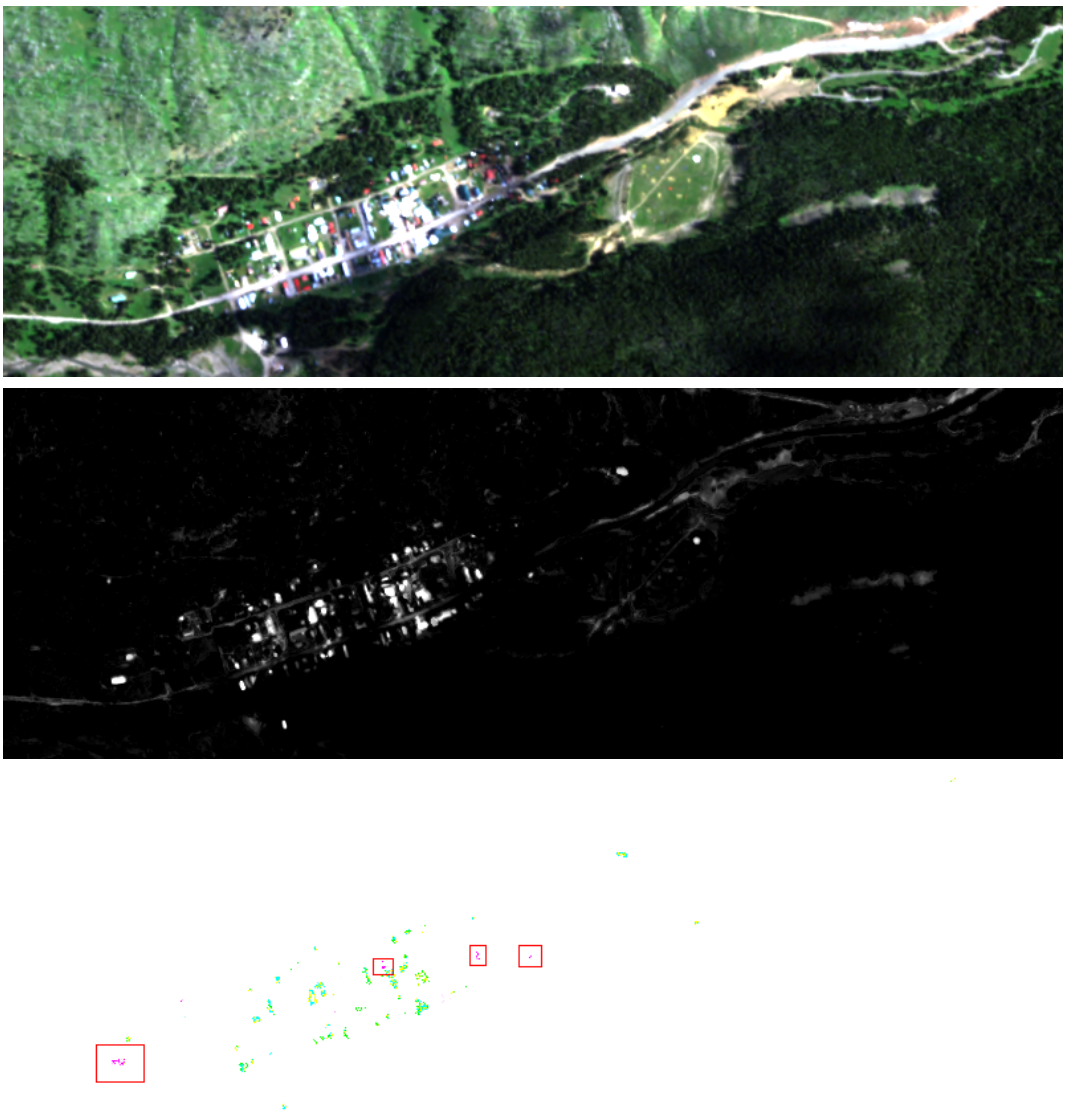


Figure 6.3: From top to bottom: RGB rendering of Cooke City, TAD rendering of Cooke City, and LLE rendering of TAD results of Cooke City with 4 areas of interest in red boxes

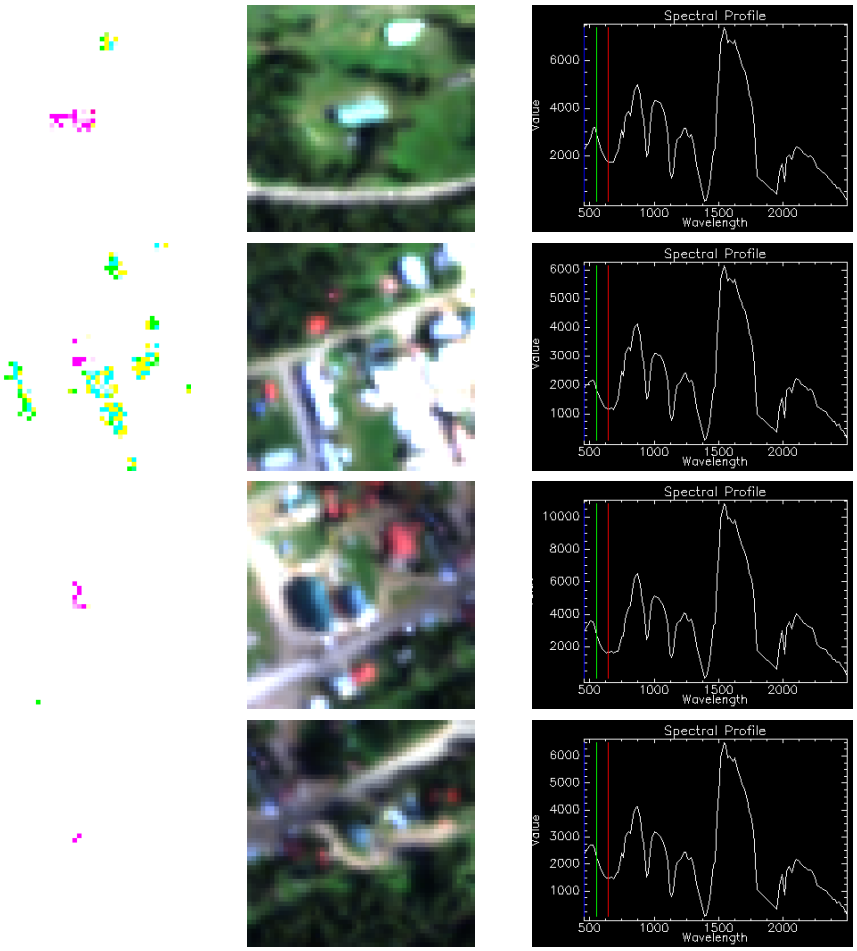


Figure 6.4: From left the right the area of interest, the RGB rendering of the same area, and a graph of the spectral vector for a pixel in the area.

Chapter 7

Conclusions

The anomaly clustering algorithm demonstrated that it can create clusters of anomalous objects that are spatially contiguous and have similar spectral signatures. It made the differentiation between point anomalies and larger anomalies clearer. The algorithm offered evidence that an algorithm that relies on combining spectral and spatial information to make clustering determination is advantageous to just spatial or spectral alone. This can facilitate the analysis of the image by giving a truer sense of the relation of anomalies that are close together. For example, it allows an analyst to see that a concentration of anomalies is actually two large objects that are composed of different materials and several other small singular pixel anomalies as we noted in the example in Chapter 4.3. Also if the analyst is looking for larger anomalies, ones that encompasses more than one pixel, such as buildings, the algorithm can reduce the number of anomalies that must be examined.

From these results, we cannot say which algorithm works more efficiently for the purposes of anomaly grouping. The anomaly grouping algorithm presented here does well at differentiating between objects of different materials and those of the same material that are not contiguous. LLE does well grouping anomalies based on their material type, but makes it hard to pick out individual objects in a group of likely constructed objects.

Chapter 8

Further Work

To further this research, we would like to improve the speed and efficiency of the algorithms mentioned, particularly the AC algorithm. There are many paths one can pursue in doing this such as the application of the QR algorithm discussed in Chapter 5.3. One possible avenue is to use faster matrix operation algorithms that rely on the symmetry inherent in the adjacency matrices. In relation to the AC algorithm we would like to add the ability for the algorithm to decide to increase or decrease the set threshold δ value in a local segment of the image. We would like to also make both the constants (δ and γ) defined by attributes of the image, not by the user, in a way that avoids the use of statistics.

Another realm of possible exploration is creating a hybrid algorithm that draws on the

strengths of the AC algorithm and LLE. Whereas LLE is able to find pixels of common materials throughout the image it loses the ability to construct anomaly groups. The AC algorithm however does a nice job of grouping pixels based on spectral and spatial similarity but cannot tell if two anomaly groupings are the same type of material. A possible application of this hybrid algorithm would be for analyzing the vehicles on an urban battle field. For instance, the algorithm would conceivably be able to pick out a set of grouped anomalies that are all tanks and another set of grouped anomalies that are all civilan vehicles.

Chapter 9

Appendix

9.1 TAD.pro

```
1  function tad_rel_prime, in1, in2
2  compile_opt idl2
3
4    a = in1
5    b = in2
6    while (b gt 0) do begin
7      t = b
8      b = a mod b
9      a = t
10   endwhile
11   res = a eq 1
12   return, res
13 end
14
15 pro tad_get_sample, fid=fid, dims=dims, pos=pos, samplesize=
       samplesize, skip=skip, base=base, offset=offset,
       to_process=to_process, $
16   pixelmask=pixelmask, cancel=cancel, samplevec=samplevec
17 compile_opt idl2
18
```

```
19    rows = long(dims[4]-dims[3]+1)
20    columns = long(dims[2]-dims[1]+1)
21    bands = n_elements(pos)
22
23    samplevec = fltarr(bands, samplesize)
24    if (skip le 10) then begin ; we want more than 10% of the
         pixels, so read a whole line at a time and pull out the
         relevant pixels
25    prev_row = -1;
26    for count = 0, samplesize - 1 do begin
27      if (n_elements(pixelmask) ne 0) then $
28          pix_offset = where(pixelmask eq (count * skip + skip)
               ) $
29        else $
30          pix_offset = count * skip + skip - 1
31        row = pix_offset / columns
32        col = pix_offset mod columns
33      if row ne prev_row then begin
34        line = envi_get_slice(fid=fid, pos=pos, line=row + dims
             [3], xs=dims[1], xe=dims[2], /bip)
35        prev_row = row
36        envi_report_stat, base, count + offset, to_process,
             cancel=cancel
37          if (cancel) then begin
38            envi_report_init, base=base, /finish
39            return
40          endif
41      endif
42      samplevec[*, count] = float(line[*, col])
43    endfor
44    endif else begin ; we want less than 10% of the pixels, so
       just read the pixels
45      for count = 0, samplesize - 1 do begin
46        if (count mod 10 eq 0) then begin
47          envi_report_stat, base, count + offset, to_process,
               cancel=cancel
48          if (cancel) then begin
49            envi_report_init, base=base, /finish
```

```
50            return
51          endif
52        endif
53        if (n_elements(pixelmask) ne 0) then $
54          pix_offset = where(pixelmask eq (count * skip + skip)
              ) $
55        else $
56          pix_offset = count * skip + skip - 1
57        row = pix_offset / columns
58        col = pix_offset mod columns
59        samplevec[*, count] = float(envi_get_slice(fid=fid, pos
            =pos, line=row + dims[3], xs=col, xe=col, /bip))
60      endfor
61    endelse
62    envi_report_stat, base, samplesize + offset, to_process,
        cancel=cancel
63    if (cancel) then begin
64      envi_report_init, base=base, /finish
65      return
66    endif
67  end
68
69
70  ; Note: All options are required
71  pro tad, fid=fid, dims=dims, pos=pos, samplesize=samplesize,
      percent=percent, out_fname=out_fname, r_fid=r_fid,
      saturate_low=saturate_low, $
72    saturate_high=saturate_high, lownorm=lownorm, highnorm=
        highnorm, normalize=normalize, m_fid=m_fid, m_pos=m_pos,
        pca=pca, clusteranomaly=clusteranomaly, lle=lle, noplot
        =noplot, bg_percent=bg_percent
73  compile_opt idl2
74
75
76
77  ;eventually make the number of pcas a user input
78    n_pcas = 12
79
```

```
80  ;;*** added by jengo *** <begin>
81    ;catch, error
82    ;if (error ne 0) then begin
83    ;     envi_error, [!error_state.msg,' ',!error_state.sys_msg
        ]
84    ;     return
85    ;endif
86  ;;*** added by jengo *** <end>
87
88    ; Defaults for some optional parameters that may not be
        there
89    if (n_elements(saturate_low) eq 0) then saturate_low = 1
90    if (n_elements(saturate_high) eq 0) then saturate_high = 10
91    if (n_elements(lownorm) eq 0) then lownorm = 1
92    if (n_elements(highnorm) eq 0) then highnorm = 2
93    if (n_elements(m_fid) eq 0) then m_fid = -1
94
95    ; Select input file and get relevant stats
96    rows = long(dims[4]-dims[3]+1)
97    columns = long(dims[2]-dims[1]+1)
98    bands = n_elements(pos)
99    sat_low = float(saturate_low) / 100
100   sat_high = 1 - float(saturate_high) / 100
101
102   envi_report_init, ['Step 1 of 2: Characterizing Background
        ...'], base=base, title='Topological Anomaly Detector (
        TAD)', /interrupt
103
104   ; If a mask is specified, determine statistics based on
        this mask.  Also, we remove rows or columns at the edges
         that are completely masked out
105   ; This costs a little bit here but saves a bunch of time/
        work later
106   orig_dims = dims
107   if (m_fid ge 0) then begin
108     pixelmask = bytarr(columns, rows);
109     for linenum = dims[3], dims[4] do begin
110       pixelmask[*, linenum - dims[3]] = byte(envi_get_slice(
```

```
            fid=m_fid, pos=m_pos, line=linenum, xs=dims[1], xe=
            dims[2]))
111       endfor
112       while (total(pixelmask[*, 0]) eq 0) do begin
113         dims[3] += 1
114         --rows
115         pixelmask = pixelmask[*, 1:rows]
116       endwhile
117       while (total(pixelmask[*, rows-1]) eq 0) do begin
118         dims[4] -= 1
119         --rows
120         pixelmask = pixelmask[*, 0:rows-1]
121       endwhile
122       while (total(pixelmask[0, *]) eq 0) do begin
123         dims[1] += 1
124         --columns
125         pixelmask = pixelmask[1:columns, *]
126       endwhile
127       while (total(pixelmask[columns-1, *]) eq 0) do begin
128         dims[2] -= 1
129         --columns
130         pixelmask = pixelmask[0:columns-1, *]
131       endwhile
132       pixels = 0L;
133       mask_inc = lonarr(columns, rows)
134       for pix = 0, rows*columns-1 do begin
135         pixels += pixelmask[pix]
136         mask_inc[pix] = long(pixelmask[pix] * pixels)
137       endfor
138       if (pixels eq rows*columns) then junk = temporary(
            mask_inc) ; undefine mask_inc
139     endif else begin
140       pixels = rows*columns
141     endelse
142
143     ; Determine the samplesize(s) to use
144     skip = long(pixels / samplesize)
145     while (~ tad_rel_prime(skip, pixels)) do --skip
```

```
146   samplesize = pixels / skip
147   if (samplesize gt 2500) then begin
148     rad_samplesize = 2500
149   rad_skip = pixels / rad_samplesize
150   while (~ tad_rel_prime(rad_skip, pixels)) do --rad_skip
151   rad_samplesize = pixels / rad_skip
152   to_process = 4 * samplesize + 3 * rad_samplesize
153   endif else begin
154     rad_samplesize = samplesize
155     rad_skip = skip
156     to_process = 5 * samplesize
157   endelse
158
159   ; Read in a sample and determine the radius.  The radius
        determination requires the pairwise distance between
        every
160   ; two samples, so we keep the samplesize small to moderate
        for this.  Thus, the radius may only be an approximate,
        but
161   ; this doesn't effect the results much
162   tad_get_sample, fid=fid, dims=dims, pos=pos, samplesize=
        rad_samplesize, skip=rad_skip, base=base, offset=0,
        to_process=to_process, cancel=cancel, $
163     samplevec=sample, pixelmask=mask_inc
164   if (cancel) then return
165   if normalize then begin
166     sample_norm = sqrt(total(sample * sample, 1, /
          preserve_type));
167     sort_index = sort(sample_norm);
168     maxnorm = sample_norm[sort_index[long(n_elements(
          sample_norm)*sat_high + 0.5)]];
169     minnorm = sample_norm[sort_index[long(n_elements(
          sample_norm)*sat_low + 0.5)]];
170     sample = TAD_Normalize(sample, minnorm, maxnorm, lownorm,
          highnorm)
171   endif
172   envi_report_stat, base, 2*rad_samplesize, to_process,
        cancel=cancel
```

```
173   if (cancel) then begin
174     envi_report_init, base=base, /finish
175     return
176   endif
177   radius = TAD_Compute_Radius(sample, percent);
178   envi_report_stat, base, 3*rad_samplesize, to_process,
        cancel=cancel
179   if (cancel) then begin
180     envi_report_init, base=base, /finish
181     return
182   endif
183
184   ; Determine the background.  If the sample used to compute
       the radius was smaller than the requested sample size,
       the whole sample is read in now
185   if samplesize ne rad_samplesize then begin
186     tad_get_sample, fid=fid, dims=dims, pos=pos, samplesize=
          samplesize, skip=skip, base=base, offset=3*
          rad_samplesize, to_process=to_process, cancel=cancel,
          $
187       samplevec=sample, pixelmask=mask_inc
188     ; we intentionally don't recompute maxnorm and minnorm,
          we want the scaling to be the same one used when
          computing the radius
189     if normalize then $
190       sample = TAD_Normalize(sample, minnorm, maxnorm,
            lownorm, highnorm)
191     envi_report_stat, base, 3*rad_samplesize + 2*samplesize,
          to_process, cancel=cancel
192     if (cancel) then begin
193       envi_report_init, base=base, /finish
194       return
195     endif
196   endif
197   sample = TAD_Separate_Background(sample, radius);
198   envi_report_stat, base, to_process, to_process, cancel=
        cancel
199   if (cancel) then begin
```

```
200       envi_report_init, base=base, /finish
201       return
202     endif
203     atria = NN_Prepare(sample);
204     envi_report_stat, base, to_process, to_process, cancel=
          cancel
205     if (cancel) then begin
206       envi_report_init, base=base, /finish
207       return
208     endif
209     envi_report_init, base=base, /finish
210     bg_percent = float(100 * n_elements(sample)) / float(bands
          * samplesize)
211     print, 'The image is estimated to contain ' + string(
          bg_percent) + ' background'
212
213     ; Now compute the rankings
214     envi_report_init, ['Step 2 of 2: Ranking Anomalies...'],
          base=base, title='Topological Anomaly Detector (TAD)', /
          interrupt
215     if (keyword_set(pca)) then result = fltarr(orig_dims[2] -
          orig_dims[1] + 1, orig_dims[4] - orig_dims[3] + 1,
          n_pcas+1) $
216       else result = fltarr(orig_dims[2] - orig_dims[1] + 1,
            orig_dims[4] - orig_dims[3] + 1, 1)
217     for linenum = dims[3],dims[4] do begin
218       envi_report_stat, base, linenum - dims[3], rows, cancel=
            cancel
219       if (cancel) then begin
220         envi_report_init, base=base, /finish
221         return
222       endif
223       line = float(envi_get_slice(fid=fid, pos=pos, line=
            linenum, xs=dims[1], xe=dims[2], /bip))
224       if normalize then line = TAD_Normalize(line, minnorm,
            maxnorm, lownorm, highnorm)
225       neighbors = NN_Search(sample, atria, line, 5)
226       if (n_elements(mask_inc) eq 0) then $
```

```
227         result[dims[1]-orig_dims[1]:dims[2]-orig_dims[1],
               linenum-orig_dims[3], 0] = float(total(neighbors
               [2:4,*], 1)) $
228       else $
229         result[dims[1]-orig_dims[1]:dims[2]-orig_dims[1],
               linenum-orig_dims[3], 0] = float(total(neighbors
               [2:4,*], 1)) * pixelmask[*, linenum-dims[3]]
230     endfor
231     envi_report_init, base=base, /finish
232     result[*, *, 0] /= max(result[*, *, 0])
233
234
235   ;;;;;;;;;;;;;;;;LLE ADDITION;;;;;;;;;;;;;;;;;;
236   if lle eq 1 then begin
237
238   k=10;nearest neighbors
239   ;numpix = rows*columns
240   ;assume mat is matrix with rows are dim values and columns
         are pixels
241   ;matcor is is matrix with rows are pixels and columns are x;y
         coordinates
242
243   ;;;;new;;;;;
244   r=0.15
245   tadresult=result
246
247   anomcount=0;
248   mat=2;
249   xycord=2;
250
251   print, 'starting to form mat'
252
253   for rownum = 0, rows -1  do begin
254   for colnum = 0, columns - 1 do begin
255
256         if result[colnum,rownum] gt r then begin
257
258           if mat[0,0] eq 2 then begin
```

```
259             mat=[envi_get_slice(fid=fid, pos=pos, line=rownum,
                    xs=colnum, xe=colnum, /bip)]
260             xycord=[colnum,rownum]
261
262           endif else begin
263             mat = [[mat],[envi_get_slice(fid=fid, pos=pos, line
                    =rownum, xs=colnum, xe=colnum, /bip)]]
264             xycord = [[xycord],[colnum,rownum]]
265
266           endelse
267
268         anomcount=anomcount+1;
269
270         endif
271
272 endfor
273 endfor
274
275
276   print, 'finished forming mat'
277
278   mat = transpose(double(double(mat)/max(mat)))
279
280   numpix=anomcount
281
282
283
284 print, size(mat)
285
286   print, 'starting step 1'
287 ;step 1
288 X_2 = double(total(double(mat^2),2)) ;vector with bands long
289 X_21 = double(fltarr( numpix,numpix));ncols,nrows
290 X_22 = double(fltarr(numpix,numpix))
291
292 ;replicating the matrix
293 for iter=0,numpix-1 do begin
294   X_21[*,iter]=double(X_2) ;rows all same
```

```
295     X_22[iter,*] = double(transpose(X_2)); cols all same
296   endfor
297
298   X_23=2*(double(transpose(mat)##mat))
299
300   distance = X_21 + X_22 - X_23 ;bandsxbands
301
302   index = fltarr(numpix,numpix) ;bandsxbands
303
304   for iter=0,numpix-1 do begin
305     index[iter,*]=sort(distance[iter,*])
306   endfor
307
308   neighborhood = index[*,1:k]
309
310    print, 'finishing step 1'
311    print, 'starting step 2'
312   ;step 2
313
314  W=fltarr(numpix,k)
315
316
317   for iter=0,numpix-1 do begin
318
319   temp = double(fltarr(k,bands))
320
321   for iterj=0,k-1 do begin
322
323   temp[iterj,*]=double(mat[iter,*]) ;cols all same hold pix
        values for iter band cols=k
324
325   endfor
326
327   z= double(mat[neighborhood[iter,*],*]-temp)
328   C=transpose(z)##z
329   ones = replicate(1,k,1)
330   W[iter,*]= invert(C)##ones
331   W[iter,*]=W[iter,*]/(total(W[iter,*]))
```

```
332 endfor
333
334    print, 'finishing step 2'
335    print, 'starting, step 3'
336
337 ;step 3
338 M=fltarr(numpix,numpix)
339
340 for iter=0,numpix-1 do begin
341
342 ww=W[iter,*]
343 jj=neighborhood[iter,*]
344 M[jj,iter] = M[jj,iter] - transpose(ww)
345 M[iter,jj] = M[iter,jj] - ww
346 M[jj,jj] = M[jj,jj] + ww##transpose(ww)
347 endfor
348
349 print, 'finishing step 3'
350
351 ;embedding
352
353 print, 'starting to find eigenvectors'
354 eigenvalues = EIGENQL(double(M), EIGENVECTORS = evecs,
       RESIDUAL = residual)
355
356 positive = where(eigenvalues gt 0)
357 positive = reverse(positive)
358
359 ev = evecs[*,positive[0:2]]
360
361 print, 'finishing finding eigenvectors'
362
363 print, 'starting reforming'
364
365 lleresult = fltarr(columns, rows, 3)
366 for iter =0, numpix-1 do begin
367
368 lleresult[xycord[0,iter],xycord[1,iter],*]=ev[iter,*]
```

```
369
370 endfor
371
372 print, 'finishing reforming'
373
374 envi_enter_data, lleresult
375
376 endif
377
378 ;;;;;;Anomaly Clustering Algorithm Addition;;;;;;;
379
380 if clusteranomaly eq 1 then begin
381 level = float(.125)
382 counter = float(2)
383 oldresult = result
384 gridsize=1
385
386 ;we will make this a 10x10 grid why not
387
388 for bigCol=0, gridsize-1 do begin
389 for bigRow=0, gridsize-1 do begin
390
391 print,'runnning submatrix' + '(' + string(bigCol) + ',' +
      string(bigRow) + ')'
392
393 if bigCol eq 0 then begin
394 colstart = 0
395 colend = (bigCol+1)*(columns/gridsize)
396 endif else begin
397 if bigCol eq gridsize-1 then begin
398 colstart = bigCol*(columns/gridsize)-1
399 colend = (bigCol+1)*(columns/gridsize)-1
400 endif else begin
401 c
402 olstart = bigCol*(columns/gridsize)-1
403 colend = (bigCol+1)*(columns/gridsize)
404 endelse
405 endelse
```

```
406
407 if bigRow eq 0 then begin
408 rowstart = 0
409 rowend = (bigRow+1)*(rows/gridsize)
410 endif else begin
411 if bigRow eq gridsize-1 then begin
412 rowstart = bigRow*(rows/gridsize)-1
413 rowend = (bigRow+1)*(rows/gridsize)-1
414 endif else begin
415 rowstart = bigRow*(rows/gridsize) - 1
416 rowend = (bigRow+1)*(rows/gridsize)
417 endelse
418 endelse
419
420 smallMatrix = oldresult[colstart:colend-1,rowstart:rowend-1]
421
422 t=0 ; number of anomolies
423
424 ;counts anomolies - not sure if we need this
425 for i = 0, rows/gridsize - 1 do begin
426 for j = 0, columns/gridsize -1 do begin
427 if ( float(smallMatrix[j,i]) gt level )  then begin
428 t = t+1
429 endif
430 endfor
431 endfor
432
433 print, 'found that there were ' + string(t) + ' anomolous
      pixels to include'
434
435 ;makes arrays
436 if t > 0 then begin
437 dect = fltarr(t,t) ;anomolies by anomiles matrix
438 dectpos = fltarr(2,t) ;the positions of the anomiles in the
      big picture
439 dectcount = 0
440
441 ;records anomolies
```

```
442 for i = 0, rows/gridsize - 1 do begin
443   for j = 0, columns/gridsize -1 do begin
444      if ( float(smallMatrix[j,i]) gt level )  then begin
445       dectpos[0,dectcount] = bigCol*(columns/gridsize) + j
446       dectpos[1,dectcount] = bigRow*(rows/gridsize) + i
447       dectcount = dectcount + 1
448       endif
449
450    endfor
451 endfor
452
453 ;print, 'found anomolies'
454
455 ;creates detection matrix - the 1 step neighbors matrix
456 for i = 0, t-1 do begin
457   dect[i,i]=1
458   for j = i+1, t-1 do begin
459
460   if((dectpos[0,i] eq dectpos[0,j] AND ABS(dectpos[1,i] -
        dectpos[1,j]) eq 1) OR (dectpos[1,i] eq dectpos[1,j] AND
       ABS(dectpos[0,i] - dectpos[0,j]) eq 1) ) then begin
461         line1 = envi_get_slice(fid=fid, pos=pos, line=dectpos
              [1,i], xs = dectpos[0,i], xe = dectpos[0,i] , /bip
              )
462         line2 = envi_get_slice(fid=fid, pos=pos, line=dectpos
              [1,j], xs = dectpos[0,j], xe = dectpos[0,j] , /bip
              )
463         dot = line1##Transpose(line2)
464         mag1 = SQRT(line1##Transpose(line1))
465         mag2 = SQRT(line2##Transpose(line2))
466         if( ACOS(dot/(mag1*mag2))*(!PI/180D) le .001D ) then
              begin
467           dect[i,j] = 1
468           dect[j,i] = 1
469         endif
470       endif
471     ;endif
472   endfor
```

```
473 endfor
474
475
476 ;performs dect + dect^2 + dect^3 + ..... until there is no
      change
477 olddect=dect
478 boolean = 0
479  for i=0, t-1 do begin
480  print,string(i) + 'of a possible' + string(t)
481  if (boolean eq 0) then begin
482    decttmp = dect ## olddect
483     for j=0, t-1 do begin
484       for k=0, t-1 do begin
485         if decttmp[j,k] ne 0 then decttmp[j,k] = 1
486       endfor
487     endfor
488     tmp = decttmp - olddect
489     boolean = 1
490     for j=0, t-1 do begin
491       for k=0, t-1 do begin
492         if tmp[j,k] ne 0 then boolean = 0
493       endfor
494     endfor
495  ;print, 'iterating matrix until false = ' + string(boolean)
496     olddect=decttmp
497     endif
498  endfor
499
500 ;why??
501 dect=olddect
502
503 ;groups anomalies - iterates through anomalies in order of
      dectpos (left->right up->down)
504 for i = 0, t-1 do begin
505   rowsum=0
506   if dect[0,i] ne -1 then begin ;check for redundency
507     for isub=0, t-1 do begin ;sees if there is more than one
        anomaly in this cluster
```

```
508         rowsum = rowsum + dect[isub,i]
509       endfor
510
511       ;we need to check if any pixels have already been labeled
              since we would want these in the same group
512
513
514       if rowsum gt 1 then begin
515
516        splitcount = 0
517        splitcounttrue=0
518        for isub=0, t-1 do begin
519           if dect[isub,i] eq 1 AND result[ FIX(dectpos[0,isub])
                , FIX(dectpos[1,isub])] gt 1 then begin
520           if splitcounttrue eq 0 then begin
521           splitcounttrue = 1
522           splitcount = result[ FIX(dectpos[0,isub]), FIX(
                 dectpos[1,isub])]
523           endif else begin
524           results[where(results eq result[ FIX(dectpos[0,isub])
                , FIX(dectpos[1,isub])])]=splitcount
525           endelse
526           endif
527       endfor
528
529         if splitcounttrue eq 1 then begin
530         result[ FIX(dectpos[0,i]), FIX(dectpos[1,i])] =
               splitcount
531         endif else begin
532         result[ FIX(dectpos[0,i]), FIX(dectpos[1,i])] = counter
533         endelse
534
535         for isub=0, t-1 do begin    ;maybe isub=i+1????
536           if dect[isub,i] eq 1 then begin
537             dect[0,isub]=-1
538             result[ FIX(dectpos[0,isub]), FIX(dectpos[1,isub])]
                    = result[ FIX(dectpos[0,i]), FIX(dectpos[1,i])]
539           endif
```

```
540         endfor
541         counter = counter + 1
542       ;   print, string(counter)
543       endif
544     endif
545 endfor
546 endif
547
548 ;outside loop for subdivisions
549 endfor
550 endfor
551
552 ;scaling the results
553
554 for c=0, columns-1 do begin
555 for r=0, rows-1 do begin
556   if float(result[c,r]) le float(1) then begin
557     result[c,r] = 0
558     endif else begin
559    result[c,r] = float(float(result[c,r])/float(counter))
560    endelse
561 endfor
562 endfor
563
564
565   bnames = 'TAD Anomaly Clustering Result'
566     out_bands = 1
567     def_bands = 0
568       envi_enter_data, result, bnames=bnames, def_stretch=
          stretch, file_type=0, inherit=inherit, r_fid=r_fid,
          def_bands=def_bands
569 print, counter
570 endif
571
572 ;;;;;;;BACK TO TADPCA;;;;;;;;;;
573
574 if lle eq 0 AND clusteranomaly eq 0 then begin
575
```

```
576     ; This is used to compute the default stretch, and also to
            build an anomaly mask if pca is being called
577     hist = histogram(result[dims[1]-orig_dims[1]:dims[2]-
            orig_dims[1],dims[3]-orig_dims[3]:dims[4]-orig_dims
            [3],0], nbins=16384, omin=dmin, omax=dmax)
578     nbins = n_elements(hist)
579     binsize = (dmax - dmin) / (nbins - 1)
580     tot_hist = total(hist)
581     cum_hist = fltarr(nbins)
582     for k = 0,nbins-1 do cum_hist[k] = total(hist[0:k]) /
            tot_hist
583
584     ; Run PCA, if requested, to colorize the results
585     if keyword_set(pca) then begin
586       ; Just run PCA on the anomalies.  Of course, if the
              degenerate case occurs where there are no anomalies,
              then run PCA on the whole image
587     ;bg_percent = 100.0
588       if bg_percent lt 100.0 then begin
589         tmp_bg_percent = 1 - (1 - bg_percent / 100.0) * pixels
                / (rows *  columns) ; if pixels were masked out,
                this adjusts things so the percentage comes out
                right.
590
```

```
591         diff = cum_hist - tmp_bg_percent
592         index_high = where(diff ge 0, num_match)
593         index_high = index_high[0]
594         index_low = max([0,index_high - 1])
595         thresh = dmin + binsize*(index_low + (tmp_bg_percent -
              cum_hist[index_low]) / (cum_hist[index_high] -
              cum_hist[index_low]))
596
597         envi_file_query, fid, ns=ns, nl=nl, xstart=xs, ystart=
              ys
598         sz = size(result, /dimensions)
599         mask = bytarr(ns,nl)
600         index = where(result[*,*,0] ge thresh)
601         ind = array_indices([sz[0],sz[1]], index, /dimensions)
602         for i = 0,n_elements(ind[0,*])-1 do mask[orig_dims[1]+
              ind[0,i],orig_dims[3]+ind[1,i]] = 1
603         envi_enter_data, mask, r_fid=mask_fid
604         envi_doit, 'envi_stats_doit', fid=fid, pos=pos, dims=
              dims, mean=avg, eval=eval, evec=evec, comp_flag=5,
              m_fid=mask_fid, m_pos=[0]
605       endif else if n_elements(mask_inc) ne 0 then begin
606         envi_doit, 'envi_stats_doit', fid=fid, pos=pos, dims=
              dims, mean=avg, eval=eval, evec=evec, comp_flag=5,
              m_fid=m_fid, m_pos=m_pos
607       endif else begin
608         envi_doit, 'envi_stats_doit', fid=fid, pos=pos, dims=
              dims, mean=avg, eval=eval, evec=evec, comp_flag=5
609       endelse
610
611       envi_doit, 'pc_rotate', fid=fid, pos=pos, dims=dims, mean
```

```
        =avg, eval=eval, $
612       evec=evec, out_dt=4, /in_memory, out_nb=n_pcas, r_fid
            =pca_fid, /forward, /noplot;, m_fid=mask_fid,
            m_pos=[0], mask_val=0
613     ;if bg_percent lt 100.0 then envi_file_mng, id=mask_fid,
          /remove, /delete
614
615   ;**** from Basener - This does a 2% stretch on the PCA
        bands ******
616   envi_doit, 'stretch_doit', fid=pca_fid, pos
        =[0,1,2,3,4,5,6,7,8,9,10,11], dims=dims, method=1, /
        in_memory, i_min=0.5, i_max=99.5, range_by=0, out_min=0,
         out_max=1, out_dt=4, r_fid=pcastretch_fid
617   for linenum = dims[3], dims[4] do begin
618       pca_line = envi_get_slice(fid=pcastretch_fid, line=
            linenum-dims[3], /bil)
619       ;pca_color = pca_line[*,0:2] + (
            [[0.5,0.5,0],[0,0.5,0.5],[0.5,0,0.5]] ## pca_line
            [*,3:5] )
620
621   ;Normalize so the total PCA brightness over the 12 bands is
        eaul to the TAD brightness.
622       pca_color = pca_line
623     result[dims[1]-orig_dims[1]:dims[2]-orig_dims[1], linenum
          -orig_dims[3], 1:12] = pca_color / (max(pca_color,
          dimension=2) # replicate(1, n_pcas)) $
624       * (result[dims[1]-orig_dims[1]:dims[2]-orig_dims[1],
            linenum-orig_dims[3], 0] # replicate(1, n_pcas))
625
626     endfor
627     envi_file_mng, id=pca_fid, /remove, /delete
628     envi_file_mng, id=pcastretch_fid, /remove, /delete
629
630     ;*** from Basener - This attemps to pick good default pca
          bands to optimise spacial discrimination
631     pcas = result[*,*,1:n_pcas]
632
633   ;Threshold PC bands to include only top 2% and save as
```

```
        PC_Spacial_Vectors
634     G = lon64arr(1000)
635     H = lon64arr(1000)
636     PC = fltarr(columns,rows)
637     PC_Spactial_Vectors = fltarr(columns,rows,n_pcas)
638     for i=0,(n_pcas-1) do begin
639       PC = reform(result[*,*,i])
640       H = histogram(PC, nbins=1000, min=min(PC), max=max(PC))
641       for j=0,999 do G[j]=total(H[0:j])
642         threshold = min(where(G gt rows*columns*bg_percent/100)
              )/1000.0
643         index = where(PC ge threshold)
644         size_index = size(index)
645         if (size_index[1] EQ 3) then index = where(PC ge (max(
              PC)/2)) ;this is to avoid errors of the threshold is
                too low.
646         PCout = dblarr(columns,rows)
647         PCout[index] = PC[index]
648         PC_Spactial_Vectors[*,*,i] = PCout
649     endfor
650
651     ;Find the PCA band that is the most spatially different
          from the others.  This will be the first (ie red) PCA
          band.
652       D = fltarr(n_pcas,n_pcas)
653       if (total(WHERE(FINITE(result, /NAN))) NE -1) then result
            [WHERE(FINITE(result, /NAN))] = 0
654       for i=1,n_pcas do begin
655         for j=1,n_pcas do begin
656           D[i-1,j-1]=total(abs(PC_Spactial_Vectors[*,*,i-1]-
                PC_Spactial_Vectors[*,*,j-1]))
657         endfor
658       endfor
659       print, total(D,1)/max(total(D,1))
660       srt = reverse(sort(total(D,1)/max(total(D,1))))
661       print, srt
662       def_bands=intarr(1,3)
663       def_bands[0] = srt[0]
```

```
664
665   ;Project PCA bands perpindicular to the first and pick a
          second band (ie green) that is most spatially different
          from the others. (Perhaps we should make the second one
          most different from the first, not the others?)
666     projected_pca = fltarr(columns,rows,n_pcas)
667     for i=1,n_pcas do begin
668       projected_pca[*,*,i-1] = PC_Spactial_Vectors[*,*,i-1] -
              PC_Spactial_Vectors[*,*,srt[0]]*(total(transpose(
            PC_Spactial_Vectors[*,*,i-1]) # PC_Spactial_Vectors
            [*,*,srt[0]])/ $
669                 (total(transpose(PC_Spactial_Vectors[*,*,i
                      -1]) # PC_Spactial_Vectors[*,*,i-1])))
670     endfor
671     if (total(WHERE(FINITE(projected_pca, /NAN))) NE -1) then
          projected_pca[WHERE(FINITE(projected_pca, /NAN))] = 0
672     for i=0,(n_pcas-1) do begin
673       for j=0,(n_pcas-1) do begin
674         D[i,j]=total(abs(projected_pca[*,*,i]-projected_pca
                [*,*,j]))
675       endfor
676     endfor
677     print, total(D,1)/max(total(D,1))
678     srt_projected = reverse(sort(total(D,1)/max(total(D,1))))
679     print, srt_projected
680     def_bands[1] = srt_projected[0]
681
682   ;Project PCA bands perpindicular to the second and pick a
          second band (ie green) that is most spatially different
          from the others.  (Perhaps we should make the third one
          most different from the second, not the others?)
683     projected2_pca = fltarr(columns,rows,n_pcas)
684     for i=0,(n_pcas-1) do begin
685       projected2_pca[*,*,i] = projected_pca[*,*,i] -
            projected_pca[*,*,srt_projected[0]]*(total(transpose
            (projected_pca[*,*,i]) # projected_pca[*,*,
            srt_projected[0]]) $
686         /(total(transpose(projected_pca[*,*,i]) #
```

```
            projected_pca[*,*,i])))
687       endfor
688       if (total(WHERE(FINITE(projected2_pca, /NAN))) NE -1)
             then projected2_pca[WHERE(FINITE(projected2_pca, /NAN)
             )] = 0
689       for i=0,(n_pcas-1) do begin
690         for j=0,(n_pcas-1) do begin
691           D[i,j]=total(abs(projected2_pca[*,*,i]-projected2_pca
                [*,*,j]))
692         endfor
693       endfor
694       print, total(D,1)/max(total(D,1))
695       srt_projected2 = reverse(sort(total(D,1)/max(total(D,1)))
             )
696       print, srt_projected2
697       def_bands[2] = srt_projected2[0]
698       def_bands = def_bands+1
699       print, def_bands
700
701     ; Save the result
702     halfway = 1.0 - float(pixels) / float(2 * rows * columns)
703     diff = cum_hist - halfway
704     index_high = where(diff ge 0, num_match)
705     index_high = index_high[0]
706     index_low = max([0,index_high - 1])
707     minstretch = dmin + binsize*(index_low + (halfway -
          cum_hist[index_low]) / (cum_hist[index_high] - cum_hist[
          index_low]))
708     maxstretch = minstretch > 0.5
709     stretch = envi_default_stretch_create(/linear, val1=
          minstretch, val2=maxstretch)
710     inherit = envi_set_inheritance(fid, dims, /spatial)
711     if keyword_set(pca) then begin
712       bnames = ['TAD Result', 'TAD PCA (R)', 'TAD PCA (G)', '
             TAD PCA (B)', 'TAD PCA (R2)', 'TAD PCA (G2)', 'TAD PCA
             (B2)', 'TAD PCA (R3)', 'TAD PCA (G3)', 'TAD PCA (B3)'
             , 'TAD PCA (R4)', 'TAD PCA (G4)', 'TAD PCA (B4)']
713       out_bands = n_pcas+1
```

```
714     endif else begin
715       bnames = 'TAD_Result'
716       out_bands = 1
717       def_bands = 0
718     endelse
719     if keyword_set(noplot) then junk = temporary(def_bands) ;
          undefines def_bands
720     if n_elements(out_fname) gt 0 then begin
721       openw, wid, out_fname, /get_lun
722       writeu, wid, result
723       free_lun, wid
724       envi_setup_head, bnames=bnames, data_type=4, fname=
            out_fname, inherit=inherit, def_bands=def_bands,
            def_stretch=stretch, file_type=0, interleave=0, $
725         nb=out_bands, ns=orig_dims[2]-orig_dims[1]+1, nl=
              orig_dims[4]-orig_dims[3]+1, offset=0, /open, /write
              , r_fid=r_fid
726     endif else begin
727         envi_enter_data, result, bnames=bnames, def_stretch=
              stretch, file_type=0, inherit=inherit, r_fid=r_fid,
              def_bands=def_bands
728     endelse
729   end
730
731   endif
732   end
```

9.2 TADGUI.pro

```
1
2   pro tad_gui_define_buttons, buttonInfo
3   compile_opt idl2
4
5     envi_define_menu_button, buttonInfo, event_pro='tad_gui',
        uvalue='none', $
6         position='after', ref_value='RX_Anomaly_Detection_', /
            sibling, $
7         value='Topological_Anomaly_Detector_(TAD)'
```

```
8
9  end
10
11 pro tad_gui, ev
12 compile_opt idl2
13
14    ; Select input file and get relevant stats
15    envi_select, fid=fid, dims=dims, pos=pos, title='Select␣
          Input␣File', /mask, m_fid=m_fid, m_pos=m_pos
16    if (fid[0] eq -1) then return
17    rows = long(dims[4]-dims[3]+1)
18    columns = long(dims[2]-dims[1]+1)
19    pixels = rows * columns
20
21    ; TAD parameters and output file selection
22    base = widget_auto_base(title='TAD␣Parameters')
23    s1 = widget_base(base, /column, /frame)
24    s2 = widget_base(s1, /row)
25    param1 = widget_param(s2, prompt='Sample␣size:␣', /auto,
          floor=250, default=1000, ceil=pixels, dt=13, uvalue='ss'
          )
26    param2 = widget_param(s2, prompt='␣Include␣edges:␣', /auto,
           default=10.0, floor=0.5, ceil=50.0, dt=4, $
27          /percent, uvalue='edges', field=1, increment=1., xsize
              =5)
28
29    s2 = widget_base(s1, /row)
30    param3 = widget_menu(s2, /auto, /exclusive, prompt='
          Colorize␣with␣PCA␣', list=['No', 'Yes'], default_ptr=0,
          uvalue='pca')
31
32    s2 = widget_base(s1, /row)
33    param5 = widget_menu(s2, /auto, /exclusive, prompt='Cluster
          ␣Anomalies␣', list=['No', 'Yes'], default_ptr=0, uvalue=
          'clusteranomaly')
34
35    s2 = widget_base(s1, /row)
36    param6 = widget_menu(s2, /auto, /exclusive, prompt='LLE␣',
```

```
        list=['No', 'Yes'], default_ptr=1, uvalue='lle')
37
38
39      s2 = widget_base(s1, /row)
40      param4 = widget_menu(s2, /auto, /exclusive, prompt='Specify
           advanced options ', list=['No', 'Yes'], default_ptr=0,
           uvalue='advanced')
41      s1 = widget_base(base, /column, /frame)
42      woutf = widget_outfm(s1, /auto, prompt='Enter output
           filename', uvalue='out_fname')
43      result = auto_wid_mng(base)
44      if (result.accept eq 0) then return
45      if ~result.out_fname.in_memory then out_fname = result.
           out_fname.name
46      samplesize = long(result.ss)
47      percent = result.edges
48      pca = result.pca
49      clusteranomaly=result.clusteranomaly
50      lle=result.lle
51
52      if result.advanced eq 1 then begin
53        base = widget_auto_base(title='TAD Advanced Parameters')
54        s1 = widget_base(base, /column)
55        s2 = widget_base(s1, /row)
56        param1 = widget_menu(s2, /auto, /exclusive, prompt='
             Normalize ', list=['No', 'Yes'], default_ptr=1, uvalue
             ='normalize')
57        param2 = widget_param(s2, /auto, prompt=' between ',
             floor=0, default=1, ceil=1e6, dt=4, field=1, uvalue='
             lownorm');
58        param3 = widget_param(s2, /auto, prompt=' and ', floor
             =0.01, ceil=1.1e6, default=2, dt=4, field=1, uvalue='
             highnorm')
59        s2 = widget_base(s1, /row)
60        param4 = widget_param(s2, /auto, prompt='Ignore outliers
             when normalizing: lower (%) ', floor=0, ceil=50,
             default=1, dt=4, field=1, uvalue='saturate_low');
61        param5 = widget_param(s2, /auto, prompt=', upper (%) ',
```

```
        floor=0, ceil=50, default=10, dt=4, field=1, uvalue='
        saturate_high')
62      result=auto_wid_mng(base)
63      if result.accept eq 0 then return
64      lownorm = result.lownorm
65      highnorm = result.highnorm
66      saturate_low = result.saturate_low
67      saturate_high = result.saturate_high
68      normalize = result.normalize
69      if (result.normalize eq 1) && (result.lownorm gt result.
          highnorm) then begin
70        envi_error, 'The lower bound on the normalization must
            be less than the upper bound.'
71        return
72      endif
73    endif else begin
74      lownorm = 1
75      highnorm = 2
76      saturate_low = 1
77      saturate_high = 10
78      normalize = 1
79    endelse
80
81    ; Run TAD
82    tad, fid=fid, out_fname=out_fname, samplesize=samplesize,
        percent=percent, dims=dims, pos=pos, lownorm=lownorm,
        highnorm=highnorm, $
83      saturate_low=saturate_low, saturate_high=saturate_high,
          normalize=normalize, m_fid=m_fid, m_pos=m_pos, pca=pca
          , clusteranomaly=clusteranomaly, lle=lle
84  end
```

Bibliography

[1] W. Basener, *Topological Approaches to HSI*, Presentation for: SPIE Conference, 2009.

[2] W. Basener, E. Ientilucci, and D. Messinger, *Anomaly Detection Using Topology*, Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XIII (S. Shen and P. Lewis, eds.), vol. 6565, SPIE, 2007, p. 65650J.

[3] W. Basener and D. Messinger, *Enhanced Detection and Visualization of Anomalies in Spectral Imagery*, Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XV (S. Shen and P. Lewis, eds.), vol. 7334, SPIE, 2009, p. 73341Q.

[4] G. Chartrand and L. Lesniak, *Graphs and Digraphs*, 4th ed., Champman and Hall, New York, 2005.

[5] J. Devore, *Probability and Statistics*, 6th ed., Thompson Brooks/Cole, Belmont, 2004.

[6] T. Doster, W. Basener, D. Messinger, and D. Ross, *Anomaly Clustering in Hyperspectral Images*, Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XV (S. Shen and P. Lewis, eds.), vol. 7334, SPIE, 2009, p. 73341P.

[7] D. Manolakis, D. Marden, and G. Shaw, *Hyperspectral Image Processing for Automatic Target Detection Applications*, Lincoln Laboratory Journal **14** (2003), 79–116.

[8] I. Reed and X. Yu, *Adaptive Multiple-Band CFAR Detection of an Optical Pattern with Unknown Spectral Distribution*, IEEE Transactions on Acoustics Speech and Signal Processing **38** (1990), 1760–1770.

[9] S. Roweis and L. Saul, *An Introduction of Local Linear Embedding*, Unpublished manuscript, 2001.

[10] L. Saul and S. Roweis, *Reduction by Locally Linear Embedding*, Science **209** (2000), 2323–2327.

[11] J. Schott, *Remote Sensing: The Image Chain Approach*, Oxford University Press, New York, 1997.

[12] D. Snyder, J. Kerekes, I. Fairweather, R. Crabtree, J. Shive, and S. Hager, *Development of a Web-Based Application to Evaluate Target Finding Algorithms*, Geoscience and Remote Sensing Symposium, vol. 2, IEEE International, 2008, pp. 915–918.

[13] L. Trefethen and D. Bau, *Numerical Linear Algebra*, Society of Industrial and Applied Mathematics, Philidelphia, 1997.