9-2006

Mining sequences in distributed sensors data for energy production.

John Damon Gant 1977-
University of Louisville

Mining Sequences in Distributed Sensors Data for Energy Production

By

John Damon Gant
B.S. University of Louisville, 2004

A Thesis
Submitted to the Faculty of the
University of Louisville
Speed Engineering School
in Partial Fulfillment of the Requirements
for the Professional Degree of

MASTER OF ENGINEERING

Department of Computer Engineering and Computer Science
University of Louisville

September 2006

Mining Sequences in Distributed Sensors Data for Energy Production

By

John Damon Gant
B.S. University of Louisville, 2004

A Thesis Approved on

September 2006

By the following Thesis Committee:

_____
Dr. Mehmed Kantardzic, Thesis Director

_____
Dr. Carol O'Connor Holloman

_____
Dr. Ibrahim Imam

DEDICATION

The author would like to dedicate this document and what becomes of his career to his mother, Sue Holland, father, Damon Gant, and stepfather Ben Holland. The author's inspiration is a direct result of the support provided by many along the way, and for that I am forever grateful.

ACKNOWLEDGMENT

The author would like to extend his gratitude and appreciation to his thesis director, Mehmed Kantardzic, for being patient and inspiring me with every word. The author would also like to thank his thesis committee for their time and dedication to teaching at the University of Louisville. Lastly a special thanks is extended to every member of the CECS department, as many of them have shown me how a true expert in his or her field inspire students and passes on knowledge.

ABSTRACT

Mining Sequences in Distributed Sensors Data for Energy Production

John D. Gant

September 28, 2006

Brief Overview of the Problem

The Environmental Protection Agency (EPA), a government funded agency, provides both legislative and judicial powers for emissions monitoring in the United States. The agency crafts laws based on self-made regulations to enforce companies to operate within the limits of the law resulting in environmentally safe operation. Specifically, power companies operate electric generating facilities under guidelines drawn-up and enforced by the EPA. Acid rain and other harmful factors require that electric generating facilities report hourly emissions recorded via a Supervisory Control and Data Acquisition (SCADA) system. SCADA is a control and reporting system that is present in all power plants consisting of sensors and control mechanisms that monitor all equipment within the plants. The data recorded by a SCADA system is collected by the EPA and allows them to enforce proper plant operation relating to emissions. This data includes a lot of generating unit and power plant specific details, including hourly generation. This hourly generation (termed grossunitload by the EPA) is the actual hourly average output of the generator on a per unit basis. The questions to be answered are do any of these units operate in tandem and do any of the units start, stop, or change operation as a result of another's change in generation? These types of questions will be answered for the years

of April 2002 through April 2003 for facilities that operate pipeline natural-gas-fired generating units.

Purpose of Research

The research conducted has dual uses if fruitful. First, the use of a local modeling between generating units would be highly profitable among energy traders. Betting that a plant will operate a unit based on another's current characteristics would be sensationally profitable to energy traders. This profitability is variable due to fuel type. For instance, if the price of coal is extremely high due to shortages, the value of knowing a semi-operating characteristic of two generating units is highly valuable. Second, this known characteristic can also be used in regulation and operational modeling. The second use is of great importance to government agencies. If regulatory committees can be aware of past (or current) similarities between power producers, they may be able to avoid a power struggle in a region caused by greedy traders or companies. Not considering profitable motives, the Department of Energy may use something similar to generate a model of power grid generation availability based on previous data for reliability purposes.

Type of Problem

The problem tackled within this Master's thesis is of multiple time series pattern recognition. This field is expansive and well studied, therefore the research performed will benefit from previously known techniques. The author has chosen to experiment with conventional techniques such as correlation, principal component analysis, and k-means clustering for feature and eventually pattern extraction. For the primary analysis

performed, the author chose to use a conventional sequence discovery algorithm. The sequence discovery algorithm has no prior knowledge of space limitations, therefore it searches over the entire space resulting in an expense but complete process. Prior to sequence discovery the author applies a uniform coding schema to the raw data, which is an adaption of a coding schema presented by Keogh. This coding and discovery process is deemed USD, or Uniform Sequence Discovery. The data is highly dimensional along with being extremely dynamic and sporadic with regards to magnitude. The energy market that demands power generation is profit and somewhat reliability driven. The obvious factors are more reliability based, for instance to keep system frequency at 60Hz, units may operate in an idle state resulting in a constant or very low value for a period of time (idle time). Also to avoid large frequency swings on the power grid, companies are required to be able to ramp-up a generator quickly using its spinning reserve.

Brief Review of Results

The results of this research identify common characteristics between generating units for the data tested. These characteristics are extremely obvious and useful on a generating unit level. Even though there were characteristics discovered, the data tested were very sparse. After looking at the testing dataset, the author feels that the distribution of data will follow a similar pattern regardless of the quarter examined. Regardless of the distribution, it is essential to process new data once released. If newer data are tested, as it should be for each new dataset released, the author is confident that the discovery of new characteristics is foreseeable. These updated characteristics along with historical patterns will allow traders to foresee high confidence electricity generation.

TABLE OF CONTENTS

		PAGE
DEDICATION		v
ACKNOWLEDGEMENTS		vii
ABSTRACT		vii
LIST OF TABLES		xiii
LIST OF FIGURES		xiv
CHAPTER		
1. INTRODUCTION		
1.1	Problem Explanation	17
1.2	Applicability of Problem	17
1.3	Previously Implemented Methodologies	18
2. REVIEW OF TIME SERIES ANALYSIS METHODOLOGIES		
2.1	Foundation of Analysis	22
2.2	The Time Series Domain	23
2.3	The Frequency Domain	26
2.4	Domain-Based Classification of Methodologies	29
2.5	Spectral Analysis	30
2.6	Wavelet Analysis and the Wavelet Transform	32
2.7	Pearson Correlation: Simple Cluster Recognition	38
2.8	Spearman Correlation: Simple Cluster Recognition	43
2.9	Introduction to Learning Techniques	47
2.10	Unsupervised Learning Techniques	48
2.11	Artificial Neural Networks	48

2.12 Supervised Learning Techniques 52

2.13 Linear Regression 53

2.14 Clustering Techniques 55

3. POWER GRID DATA ANALYSIS

3.1 Characteristics of System 60

3.2 Goal 61

3.3 Type of Data 62

4. DATA COLLECTION AND PREPROCESSING

4.1 Introduction 65

4.2 Discussion of Observations 66

4.3 Data Population 68

4.4 Discussion of Dimensionality 69

4.5 Does the data exhibit normality? 71

5. MODIFIED K-MOTIF DISCOVERY APPLIED TO THE POWER DOMAIN

5.1 Review of Theory 75

5.2 The Coding Process 77

5.3 K-Motif Discovery Algorithm 79

5.4 Rule Creation 81

5.5 Example of Coding, and Motif Discovery 82

6. EXPERIMENTAL RESULTS

6.1 Discussion of Data Analyzed 88

6.2 Introduction to Results 88

6.3 Unit Raw Data 90

6.4 Principal Components Analysis for Units 92

6.5 K-Means Clustering Analysis for Units 94

6.6 K-Motif Discovery for Units 96

6.7 Plant Raw Data 99

6.8 Principal Components Analysis for Plants 100

6.9 K-Means Clustering Analysis for Plants 102

6.10 K-Motif Discovery for Plants 103

6.11 Results Discussion 103

7. CONCLUSION 105

REFERENCES 109

APPENDIX 1: SOURCE CODE 115
APPENDIX 2: UNIT-LEVEL ASSOCIATION RULES 148
APPENDIX 3: PLANT-LEVEL ASSOCIATION RULES 152
CURRICULUM VITAE 153

LIST OF TABLES

PAGE

1. Table 3.1 – Methodologies and their Corresponding Domains 29
2. Table 3.2 – Commonly Used Mother Wavelets 36
3. Table 3.3 – Paradigms of Artificial Neural Networks 49
4. Table 3.4 – Distance Measures 57
5. Table 5.1 – Goodness of Fit 73
6. Table 6.1 – Coding of Raw Data 85
7. Table 7.1 – Summary of Unit-Level Cross Correlations 92
8. Table 7.2 – Summary of Statistics for Unit-Level Clustering 95
9. Table 7.3 – Sample of Discovered Sequences including Support and Confidence Values 97
10. Table 7.4 – Summary of Plant-Level Cross Correlations 99
11. Table 7.5 – Summary of Statistics for Plant-Level Clustering 102

LIST OF FIGURES

PAGE

1. Figure 3.1 – Components of the Complex Plane 27
2. Figure 3.2 – An Example Spectral Plot 32
3. Figure 3.3 – An Example Wavelet Map 38
4. Figure 3.4 – Exact Likeness Plot, High Correlation 42
5. Figure 3.5 – Exact Inverse Likeness Plot, High Negative Correlation 42
6. Figure 3.6 – Simple Neuron, component of Artificial Neural Network 50
7. Figure 3.7 – Simple Feedforward Neural Network 51
8. Figure 3.8 – Recurrent Network Example 51
9. Figure 3.9 – Example of a Linear Regression Fit 54
10. Figure 5.1 - Relative Frequency Distribution for Unit 6A at Barry Generating Station 74
11. Figure 6.1 – Sequence Discovery over Sets 83
12. Figure 6.2 – Example of Two Time Series in Raw Form 83
13. Figure 6.3 – Unit 3 Coding 84
14. Figure 6.4 – Unit 4 Coding 84
15. Figure 6.5 – Motif Discovery 85
16. Figure 7.1 – Unit-Level Plot of Learning Data Set Percentages of ValuesReported 90
17. Figure 7.2 – Unit-Level Plot of Testing Data Set Percentages of Values Reported 91
18. Figure 7.3 – Screeplot of Unit-Level Learning Dataset 92

19. Figure 7.4 – Within-Cluster Error for the Unit-Level Learning Dataset 95

20. Figure 7.5 – Screeplot for Plant-Level Learning Dataset 101

21. Figure 7.6 – Within-Cluster Error for Plant-Level Learning Dataset 102

INTRODUCTION

Problem Explanation

As noted earlier, the research for this Master's thesis is based on a time series of data from the Environmental Protection Agency. Each generating unit that has a capacity, or the theoretical maximum output, greater than 25 MWH must report to the EPA. The data reported are via a SCADA system. A SCADA system is a large-scale distributed measurement and control system [19]. SCADA consists of measurement and reporting devices that monitor operation of the generating units. The EPA uses SCADA data when assessing penalties against power companies based on emissions violations. This data are publicly available and are the source data for the research conducted within this Master's thesis. The data consist of many emissions-related factors but of importance is average hourly power generation on a per unit basis. This data are averages of all power generated for the observed hour via the generating unit. The objective is to discover characteristics between generating units while in normal operation. Normal operation includes daily cyclic patterns and ramp-up and ramp-down periods. A more basic approach is to see the analysis as a pattern-matching problem in a highly dimensional domain. Each pair of characteristic generation patterns in tandem will allow the researcher to construct an association rule that would be used by analysts in creating power modeling scenarios.

Applicability of Solution

The real-world data used in this Master's thesis, a discrete time series, are typical data faced by researchers. Specifically here the author attempts to discover common

sequences between electric generating units to be used in the field of power generation or commodity trading and find matching sequences. Along with sequence mining, the author identifies a coding schema that fits the target dataset more reliably than other proposed schemas for time series. Many researchers have studied the topic of sequence mining as it is of interest in many fields including DNA sequencing, and weather modeling [23,37,38,21,39,16,17,18,40]. With respect to coding schemas, many researchers have custom coding schemas and this thesis was no different. Coding Schemas are used to allow for normalization and abstraction of the data; moreover, they allow the algorithms to discover similarities between sequences that do not obviously exist using raw data. The most common schemas are those based loosely on Gaussian and Uniform Populations. Lastly sequence matching is attacked by a majority of time series researchers. Sequence matching can be performed on streams or historical data. Some researchers focus more on stream mining, or real-time sequence discovery, and spend most of their time researching more practical implementations that are intended to run in optimal time [23,17].

Previously Implemented Methodologies

Researchers at Harvard [21], the University of California at Riverside & Irvine [16,17,18,40], the University of California at Los Angeles [39], the University of Applied Sciences in Wolfenbüttel, Germany [38], the Université Neuchátel [37], and the Universities of Maryland & Virginia & Michigan & Carnegie Mellon along with AT&T [23] have studied discovery of temporal sequences along with coding schemas. Many of

these researchers have novel ideas involving implementation of coding and statistical discovery.

The team at Harvard expanded the technique presented by Bussemaker [41], deemed the Moby Dick motif discovery method. The Moby Dick motif discovery method can be expressed simply as determining probabilities of sequence based on their subsequence counterparts via a dictionary. These probabilities are created based on the number of possibilities for construction of the sequence based on the current contents of the dictionary, the product of probabilities of the sequence's subsequences, and the number of times the sequence has been used. Their contribution included the use of a stochastic dictionary and data augmentation. The stochastic dictionary is based loosely on the probability that a letter will occur in a word, or code, in a sequence. For each sequence there is a probability matrix, termed PWM, constructed. Each code has a derived probability based upon its location in the sequence.

The team from the University of California at Riverside have expanded on their previously released theory of discovering gene sequences based upon their codes deriving from a normal distribution, termed K-Motifs [40]. Before the introduction of the current additions, one must review the previously released material. This included the introduction of motifs and statistical representations of a time series dataset. The introduction of motifs and then K-Motifs allowed for the probabilistic coding of time series. This coding is essential is discovering useful and frequent sequences, or K-Motifs. The research in this Master's thesis is based highly on the theory introduced by this group

at University of California at Riverside. The group's current addition is one that accommodates for missing data, termed "don't cares" [16]. This discovery algorithm is novel in that is has the ability to overlook noise in data and can therefore discover based on bad data. It has advantages over the other more sophisticated algorithms, in that in does not need prior knowledge of the data nor does it use dictionary methods.

The team at the University of California at Los Angeles studied the evolution of DNA sequences built from binary matrices converted into tree structures. As a sequence is constructed it traverses the tree based upon a likelihood function. This likelihood function is based upon a probability, which comes from unvisited parents within the tree. The main premise for this theory is that the sequences are created via an immigration-death model.

The researcher at the University of Applied Sciences uses a novel technique to discover important sequences. Although he does not discover every sequence, as most do, he intends to only discover useful sequences using derivatives (as he considers sequences as changes in both the first and second derivative). Prior to calculating the derivatives of the signal a smoothing process is used to reduce noise, which could be responsible for false zero-crossings. Using a sliding window approach the researcher attempts to derive association rules.

At the Université Neuchátel researchers discretize the raw data into coded values. After discretization they generate classification trees using the C.45 algorithm. To perform the

coding schema, the researchers use difference measures between sequences. The sequences are generated from fixed length intervals, and therefore lack knowledge of sequence creation.

Finally the group consisting of researchers from AT&T, and the Universities of Maryland & Virginia & Michigan and Carnegie Mellon developed an algorithm, deemed MUSCLES, to discover correlations, outliers, and corrupt data via linear techniques. Although the authors perform their analyses on sequences, it may be classified as work over a discrete time series and not sub-sequences as most have chosen previously. Their work is directed at prediction of next values delivered by an evolving sequence or data stream.

.

REVIEW OF TIME SERIES ANALYSIS METHODOLOGIES

Foundation of Analysis

There must be a goal for an analysis, which usually starts out as a question. This question is formulated into a hypothesis, or a proposed answer to this question. To perform an efficient data mining operation, a hypothesis must be the main driver. Usually hypothesis creation is domain specific. Sometimes, in a business setting, the hypothesis is not necessarily scientifically motivated. Nevertheless the analyst must, with the help of the domain expert (which could be one in the same), from some previous observations create the hypothesis. This hypothesis is a component of the scientific process. The scientific process is defined as follows:

1. *Develop a hypothesis.* The hypothesis is the question that drives analysis. It is the why or how that is the sole reason for experimentation.
2. *Conduct the experiment.* Within this document, conducting the experiment refers to sampling data from power plants. Since this is already done, via SCADA, the only step left is to decode of the data in preparation for the analysis
3. *Analyze the data.* The majority of this document is centered on analysis of data. Analysis can include some as simple as aggregate processing up to complex pattern recognition processes. This section should be embarked upon with care as performing hastily during analysis will flaw the entire outcome.

4. *Compare the results.* Either the domain expert or the analyst can perform this step. It should include common-sense checks of the data against facts or unwritten truths. Results that are abnormal should be suspect and examined carefully. Abnormal data does not confirm faulty analysis, but could indicate distinct features that need to be recognized.
5. *Determine a conclusion.* With the help of a domain expert, the researcher may report his or her conclusion for future use.

The Time Series Domain

A time series can be defined as a set of values keyed uniquely by timestamp. If a time series, X, is

$$X = \{(T_0, V_0), (T_1, V_1) \ldots, (T_{n-1}, V_{n-1})\} \qquad (3.1)$$

then T_x represents the distinct timestamp that corresponds to the value V_x and n identifies the length of the time series. Time series data are values recorded at some point in time. Time series analysis is mathematical processing of the time series data to obtain a conclusion concerning a hypothesis. The most common time series analysis is stock market analysis. The changes associated with stock prices are recorded at tick intervals and used by traders to predict characteristics of given stocks. The ability to identify characteristics of stock prices allows traders to model, or predict, the future outcome of prices and gain a profit from this knowledge.

Time series data sets typically consist of a set of two-dimensional data pairs. This data are keyed, or uniquely sorted, by a timestamp value. The value of the pair can be either a categorical or numerical value. Categorical values are typically nouns. For example the color of the sky on a particular day is a categorical value, keyed by day. Numerical data are also a value and can consist of any of the basic numerical representations. Some examples of numerical data are fractional, imaginary, and whole numbers. Time series data are usually collected from some process that is sampled at precise time intervals. Sampling is an important concept, and can be described as listening for data over a medium and recording these values for later processing. This can be accomplished by storing data in memory, writing values down on paper, or by other means that allow someone to record values that happen at an instant in time. Most time series data sets are sampled using a rate that are generally the same for all data points sampled, resulting in a common unit of time. The elapsed time, or interval, of sampling with relation to the amount sampled is regarded as the sampling rate, or S that is defined as

$$S = \frac{V}{\Delta t} \qquad (3.2)$$

where V is the number of samples taken (or points captured) and, Δt the time elapsed between the start and end of the sampling process.

For example, if a heart monitor samples a heartbeat 1000 times in a one second interval, it can be said that this monitor has a thousand samples per second sampling rate. Use of historic time series data provide researchers the ability to view trends, or the general

direction of movement of a value. Time series data are always analyzed after sampling. Although this document concerns itself with data that was sampled years ago, that is not always the case. For instance a real time operating system might analyze data that are sampled by a sensor on an aircraft wing virtually instantaneously, allowing the operating system to adjust flight controls for autopilot. Many researchers [14,15] investigate these complex issues, and real-time data sets are considered to be data streams, i.e. continuous sampling systems.

Although a simple time series is an important concept, most likely in mining and modeling problems the researcher will be faced with multiple time series. A multiple time series matrix, X, is defined as

$$X = \begin{bmatrix} (T_0, V_0) & .. & .. & (T_{n-1}, V_0) \\ .. & .. & .. & .. \\ .. & .. & .. & .. \\ (T_0, V_{n-1}) & .. & .. & (T_{n-1}, V_{n-1}) \end{bmatrix} \qquad (3.3)$$

where each column vector of X represents a single time series. This is a typical representation for multiple time series when using linear algebra to calculate values. A linear algebra approach is common in least square calculation of linear regression.

Time series data can become stale. The meaning of stale is domain dependent, but basically refers to the length of time the data are useful after sampling. The usefulness of data can be a few seconds or decades. Storage of stale data is a great concern in business

applications. Large network storage devices handle this duty. Large data warehouses store either aggregates, mathematical summaries, of the data or the raw data itself. Storing the raw, or sampled, data are very costly and further scrutinizes the mark set for staleness. For the most part data warehouses store aggregates of the raw data. Data mining can be described as locating similarities, and drawing likeness conclusion about data. Usually, mining is performed aggregated data. If the miner has access to pre-aggregated data, as in a large warehouse and the aggregates are sufficient for the mining goal, the mining task is much easier. Choosing these aggregates is a complicated task that requires domain knowledge. Efficient storage of historic time series data is a challenging research interest, and a practical challenge for business applications.

Although time series data are easy to comprehend in small sections, most time series data are only useful when compared over large spans of time. Spotting patterns in small data sets is trivial for humans, and easy for computers. But when time series data are analyzed in extremely large sets, it becomes all but impossible for humans and challenging for computers to complete in a reasonable amount of time. The field of data mining takes this challenge upon itself.

The Frequency Domain

We have previously discussed the time series domain as the prominent domain for statistical analysis. Although the time series domain is the most common, the field of signal processing offers the frequency domain as an option for data analysis. The frequency domain allows a signal's components to be exposed. Most signals are a

combination of many small signals, which vary in power. To observe these signals one must transform the data to the frequency domain. Transforming data from one domain to another can be accomplished with both data streams (continuous) and finite data sets (discrete). Once transformed the data are no longer in a coordinate system described by unit of time and value, but now represented by coordinates in the complex plane. These coordinates consist of a phase angle and a magnitude, which are calculated via transforms.

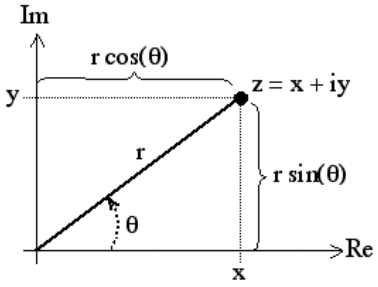

Figure 3.1 Components of the Complex Plane:

There are many transforms that exist to convert between domains, but the most common are the Laplace, Z, and Fourier transforms. The Complex Unilateral Laplace transform is defined as

$$L\{f(t)\} = \int_0^\infty e^{-st} f(t)dt \qquad (3.4)$$

$$s = \sigma + i\omega$$

where *s* represents a combination of both the real and imaginary portions of the signal. The Continuous Fourier transform otherwise known, as the Fourier transform, is similar to the Laplace transform by the fact that if for

$$s = \sigma + i\omega$$

Sigma is set to zero, $\sigma = 0$, meaning there are no real components only complex. From this it is evident that the Fourier transform takes the shape of the Laplace transform including normalization constant. The Fourier transform is defined as

$$F(t) = \frac{1}{\sqrt{2\pi}} \int_0^{\infty} e^{-i\omega t} f(t) d\omega \qquad (3.5)$$

where *e* represents the exponential function. Although the Fourier transform is continuous, its popularity eventually gave way to approximations of a discrete nature: specifically the Discrete Fourier transform, DFT, is used in frequency domain analyses. The DFT is a heavily optimized transform with years of software optimizations to improve calculation speed resulting in the Fast Fourier transform and its counterparts. The Discrete Fourier transform is defined as

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} f(t) e^{\frac{i2\pi nk}{N}} \qquad (3.6)$$

where n = 0...N-1. A discrete version of a transformation function, which is very similar to the Unilateral Laplace transform, is the Z transform. The Z transform is defined by the following equation:

$$X\{f(t)\} = \sum_{t=0}^{\infty} f(t)z^{-t} \qquad (3.7)$$

$$z = re^{i\omega}$$

or

$$z = re^{i2\pi f}$$

For the purpose of discrete analysis, choice of the Z and Discrete Fourier Transforms are appropriate. A typical representation of the Fourier transform is via a Power Density plot, i.e. Spectral Analysis.

Domain-Based Classification of Methodologies

Both the frequency and time series domains are valuable tools for the researcher. There are many methodologies for each domain, but the review below will limit itself to the most commonly used and most beneficial. Table 3.1 specifies the domain for each methodology reviewed.

Table 3.1 - Methodologies and their Corresponding Domains

Methodology	Domain
Spectral Analysis	Frequency

Wavelet Analysis	Frequency and Time
Pearson Correlation	Time
Spearman Correlation	Time
Artificial Neural Networks	Time
Regression Analysis	Time
Principal Component Analysis	Time
K-Means Clustering	Time

Spectral Analysis

To use the Fourier transform one must begin with the Discrete Fourier transform, DFT, as time series (not signals) are discrete by nature. If the researcher were concerned with the theory of continuous signals, they would ignore the DFT and focus on calculations based around the Continuous Fourier transform. Although there is many implementations of the Discrete Fourier transform, the Fast Fourier transform is the most widely used. A stipulation to using the Fast Fourier transform is the discrete data set involved must be of a power of two in size. For example if the series were of length 60 it would have to be padded, or centered by adding zeros, to a data window up to a size of 64. For more information on windowing please refer to Masters[11]. One result of this padding is that the signal at lower (left side of spectral diagram) and higher (right side of spectral diagram) will be affected. The effect is a decrease in power of the frequency represented in the spectrum. There are guidelines, which are cautionary indicators that indicate the frequency range that has been affected by the padding. Torrence and Compo [25] refer to

this restriction as the *e*-folding time, τ_s. The result of a DFT provides us with a listing of complex numbers. Each complex number in this listing must transformed into its hypotenuse by

$$hyp = \sqrt{real^2 + complex^2} \tag{3.8}$$

And from a listing of hypotenuses one may calculate the power for each frequency, or iteration in the list. The equations 3.9 allow the researcher to calculate the power for each frequency band.

$$\begin{aligned} P_0 &= \frac{|hyp_0|^2}{n^2} \\ P_i &= 2 * \frac{|hyp_i|^2}{n^2} \\ P_{\frac{n}{2}} &= \frac{\left|hyp_{\frac{n}{2}}\right|^2}{n^2} \end{aligned} \tag{3.9}$$

With power values, one may scatter plot the power against their frequency counterparts. Another effect of using Fourier analysis is the reflection of the signal. The signal reflection is observed at half of the period of the signal. To observe this power spectrum, one uses a Power Density plot. Using this power density plot a researcher may discern the dominating frequencies from the time series. Noting these dominant frequencies are crucial to removing all continuous patterns from the series, thus allowing the real data to

show. Careful note must be taken when interpreting these plots, and experience is of great value. Figure 3.2 is an example Spectral plot.

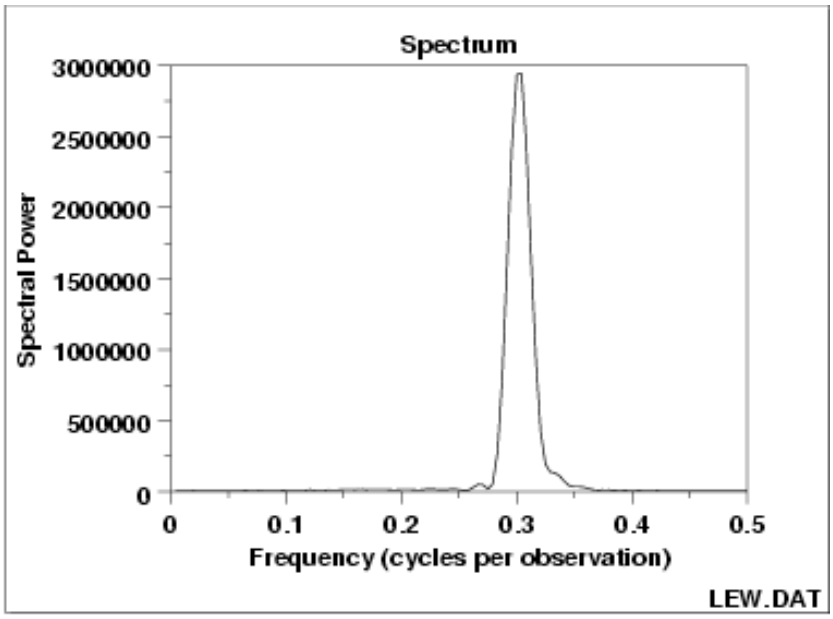

Figure 3.2 – Spectral Plot Example

The y-axis consists of spectral power values, as defined above. The x-axis represents the corresponding frequencies observed within the signal. A large value for power indicates a significant frequency present in the signal. In this example at 0.3 there seems to be a dominant component. There should be further study of the signal, including removal of the offending frequency and retesting to assure the correct frequency was identified.

Wavelet Analysis and the Wavelet Transform

The Fourier transform is essential to signal analysis, but is flawed when used with time series data sets. This difficulty is due to the inability, using DFT, to pinpoint a frequency's component in time. Having the knowledge that a 40Hz component is very

strong in a signal has no relation to its presence in time. To accomplish mapping of frequency to time, one should use wavelet analysis. Alfred Haar introduced the first wavelet in the early 20th century [26]. The noted father of wavelet theory, Morlet, in 1983 introduced the Morlet wavelet and wavelet transforms [27]. Wavelet analysis consists of using either the Continuous Wavelet transform or the Discrete Wavelet transform to decompose a time series represented as a signal. The mother wavelet defines the daughter wavelet which is the essential wavelet used during the analysis. Wavelet analysis includes choosing a mother wavelet and a real signal to represent the time series. Once a signal has been chosen, a researcher must choose a mother wavelet from which the daughter wavelets can be calculated. A mother wavelet is chosen by analysis of the base signal. Specifically, if one has a smooth signal then choosing a wavelet that resembles a pulse or spike would result in poor decomposition. The discrete wavelet transform is used only with orthogonal mother wavelets; therefore, all other mother wavelets are used via the continuous wavelet transform. Both continuous and discrete versions of the wavelet transform use daughter wavelets, which are based directly on mother wavelets. The continuous wavelet transform, $\gamma(s,t)$, is defined as

$$\gamma(s,t) = \int f(t)\psi^*_{s,t}(t)dt \qquad (3.10)$$

Where $\psi^*_{s,t}$ = Complex Conjugate of the daughter wavelet.

Its corresponding daughter wavelet (non-orthogonal) is expressed as

$$\psi_{s,t}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-\tau}{s}\right) \qquad (3.11)$$

Where

f(t) = Continuous Signal

s = Scale

τ = Position

ψ = Mother wavelet

For orthogonal-based wavelets one uses the Discrete Wavelet transform, which is defined as

$$f(t) = \sum_{j=-\infty}^{\infty}\sum_{k=-\infty}^{\infty} d_k^j \psi_k^j(t)$$
$$d_k^j = \int f(u)\psi_k^j(u)du \qquad (3.12)$$

And its counterpart the daughter wavelet (Orthogonal)

$$\psi_{s,t}(t) = \psi(2^s t - \tau) \qquad (3.13)$$

The topics of orthogonal and non-orthogonal daughter wavelets are directly related to the mother wavelets. A mother wavelet is deemed orthogonal when it can be proven that the function has an orthogonal basis, or those dot products of the vectors that make up the base which equate to scalar values. A special case of an orthogonal basis is one that equates to one and this case is deemed orthonormal. Typically orthogonal wavelets are

not used in time series analysis due to their representation, a pulse or spike, which is unlikely in a typical time series. For example the Haar wavelet is a representation of a step function. Most time series are smooth, relative to a pulse, and would not be accurately (with low residual error) represented using a Haar wavelet.

There are many mother wavelets that may be used and are defined by the admissibility and regularity conditions. The admissibility condition is defined as

$$C = \int \frac{|\gamma(t)|^2}{t} \qquad (3.14)$$
$$\gamma(\omega) = FT(\psi(t))$$

where FT is the Fourier transform of the function $\psi(t)$, a function that is tested as a mother wavelet. The implication of the admissibility condition is that the value of the Fourier transform, of the mother wavelet, disappears at zero frequency, which means there is a cutoff and no data is lost. The admissibility condition also implies that the function must act like a wave with mean zero [29].

$$\int_{-\infty}^{\infty} \psi(t)dt = \gamma(t) = 0 \qquad (3.15)$$

The regularity condition implies that for n-1 values produced by the signal, they also average to a mean of zero [29]. In literature this is described as “vanishing moments”.

$$\int_{-\infty}^{\infty} x^k \psi(t)dt = 0 \qquad \text{for k = 0 ..n-1} \qquad (3.16)$$

$$\int_{-\infty}^{\infty} x^k \psi(t)dt \neq 0 \qquad \text{for k = n}$$

If a wavelet satisfies both conditions it can be considered to be a mother wavelet and will perform correctly using wavelet transformation. Table 3.2 contains a listing of common mother wavelets.

Table 3.2 - Commonly Used Mother Wavelets

Wavelet Name	Description
Haar	$\psi(t) = \begin{cases} 1 & 0 \le t < \frac{1}{2} \\ -1 & \frac{1}{2} \le t < 0 \\ 0 & otherwise \end{cases}$
Hermitian	$\psi(t) = (2n)^{-\frac{n}{2}} C_n H_n\left(\frac{t}{\sqrt{n}}\right) e^{-\frac{1}{2n}t^2}$ $C_n = \left(n^{\frac{1}{2}-n}\Gamma\left(n+\frac{1}{2}\right)\right)^{-\frac{1}{2}}$ $where \quad \Gamma(n) = (n-1)!$ $H_n = (-1)^n e^{\frac{x^2}{2}} \frac{d^n}{dx^n} e^{-\frac{x^2}{2}}$
Morlet	$\psi(t) = C_\sigma \varpi^{-\frac{1}{4}} e^{-\frac{1}{2}t^2} \left(e^{i\sigma t} - k_\sigma\right)$ $k_\sigma = e^{-\frac{1}{2}\sigma^2}$ $C_\sigma = \left(1 + e^{-\sigma^2} - 2e^{-\frac{3}{4}\sigma^2}\right)^{-\frac{1}{2}}$

Before the daughter wavelet can be used within the wavelet transform, it must be scaled. Scaling is useful for its ability to transform non-orthogonal wavelets into wavelets with an orthogonal base. The energy of a daughter wavelet must be scaled, so that it may be compared to other wavelets. For more information on scaling of wavelet refer to Torrence and Compo [25] and Mallat[42].

In practice wavelet transformation allows a researcher the ability to pinpoint in time the location of frequency components, or localization of the frequency components. Fourier analysis gives a global overview of composing frequencies, but Wavelet transforms allow the researcher the ability to view both global and local characteristics. Wavelet transformations are used heavily in geophysics, image processing, and audio processing. Figure 3.3 is a typical representation from a wavelet transform in the form of a wavelet map.

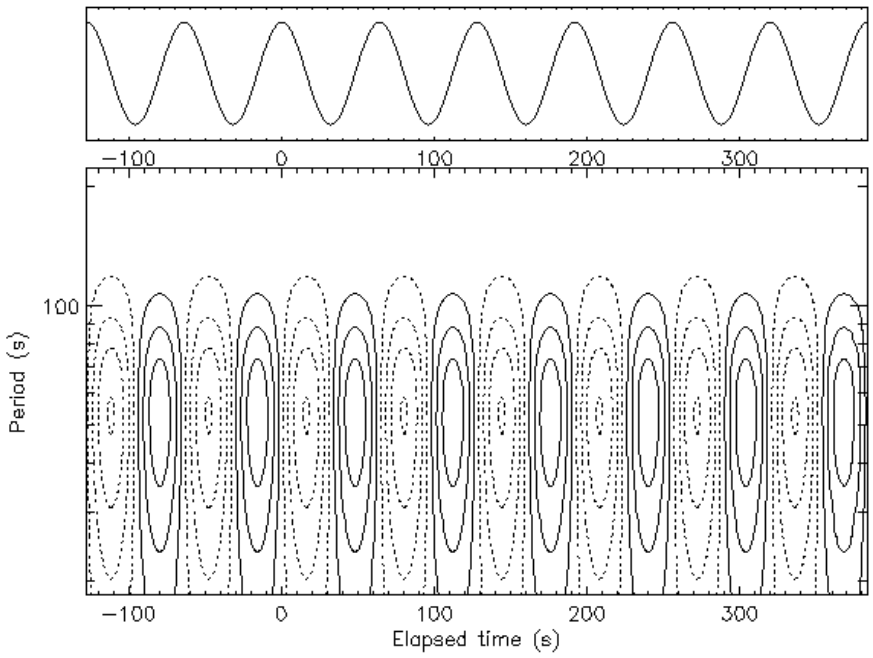

Figure 3.3 - Wavelet Map Example

Pearson Correlation: Simple Parametric Cluster Recognition

Within the field of statistical inference there exist two domains, hypothesis testing and point estimation. This Master's thesis is primarily concerned with point estimation. The author recognizes that hypothesis testing is an element within model validation along with being a subfield of statistical inference. To estimate a point, usually a statistical model is developed on previous data. Statistical model construction will be discussed later but now it must be stated that there are independent elements within a statistical model, and clustering allows a researcher the ability to 'identify' these elements. As said previously, pattern recognition, and in the context of this document, clustering is an easy task for the human eye. Although the human eye does an excellent job of spotting clusters in two-dimensional space, in spaces greater than two-dimensions algorithms provide a

more thorough and trustworthy source for cluster recognition. It must be stated that even though the human eye does not perform as well as algorithms in highly dimensional space, it is the trustworthy tool used to check the accuracy of algorithms.

Simple numerical clustering can be tackled by techniques borrowed from the field of statistics. The sum of squares is a measure of error. The sum of squares is used due to its ease of derivation, which makes for cleaner mathematics. There are multiple forms of the sum of squares operation, but in general using the arithmetic mean is most common.

Arbitrary list, L, of points

$$L = \{p_0, p_1, p_2 ....., p_n\}$$

Sum of squares of L,

$$SS_L = \sum_{i=0}^{n} (p_i - \bar{p})^2 \qquad (3.17)$$

Where the sample arithmetic mean,

$$\bar{p} = \frac{\sum_{i=0}^{n} p_i}{n} \qquad (3.18)$$

Simply put, the sum of squares is a measure of variability within a list. An important note is that an outlier can easily skew the error function. An outlier can be classified as a data point that is 'abnormal' with respect to the data list as a whole. An outlier is usually domain specific, but can be associated with the list's sample standard deviation. Simply it can be any data point that falls outside a multiple of the list's sample standard deviation. Outliers result in a misleading sum of squares for a particular list of data that contains the large outlier. The sum of squares is a common measure used in many areas of statistics, including Analysis of Variance (ANOVA) and correlation analysis. When analyzing data for similarities a simple measure is the sample correlation, also known as the Pearson Correlation coefficient.

$$r = \frac{SS_{xy}}{\sqrt{SS_x SS_y}} \tag{3.19}$$

Where appropriate sum of squares are defined by

$$SS_{xy} = \sum_{i=0}^{n} (x_i - \bar{x})(y_i - \bar{y}) \qquad SS_x = \sum_{i=0}^{n} (x_i - \bar{x})^2 \qquad SS_y = \sum_{i=0}^{n} (y_i - \bar{y})^2 \tag{3.20}$$

Pearson Correlation coefficient is a linear measurement of likeness. A linear measurement of likeness is one that attempts to fit a line through a set of points based upon the mean. Typically the square value of the correlation coefficient is used in measuring likeness between series. This coefficient, P, is referred to as r-squared and is defined by:

$$P = r^2$$

Regardless of the type of correlation used, it is still bounded by lower and upper limits. There are many types of likeness measures and from that many types of correlations. The types of correlation include parametric, where the data comes from a know probability distribution, and non-parametric, where the distribution of that data is unknown. Correlation is easily abused, when the assumption of a known distribution is either assumed or ignored entirely. Correlation is used to establish likeness between data lists. A correlation of positive one, establishes an exact likeness, conversely a correlation of negative one identifies an exact inverse likeness.

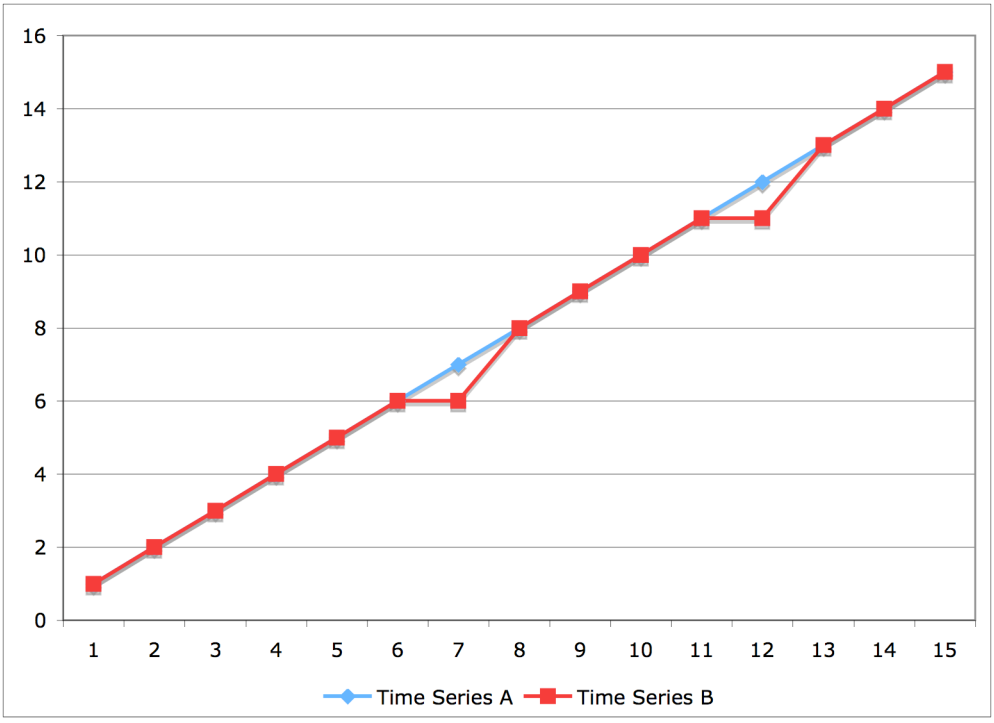

Figure 3.4 - Exact Likeness, exhibits High Correlation

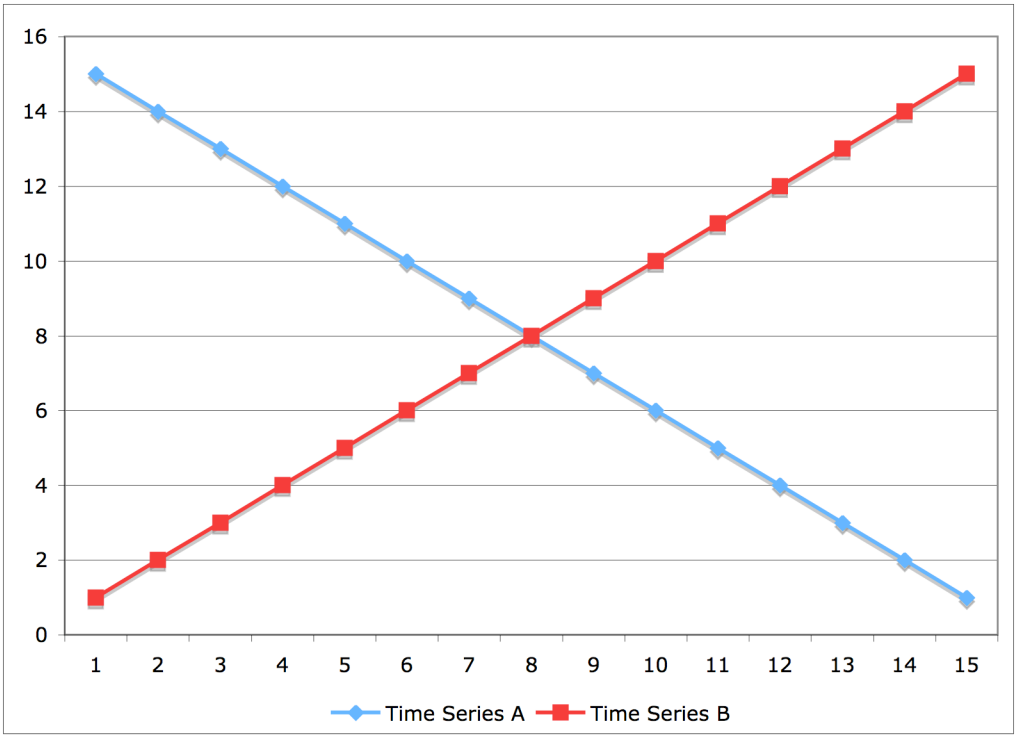

Figure 3.5 - Exact Inverse Likeness, exhibits High Inverse Correlation

As shown in figures 3.4 and 3.5, correlation is not an absolute measure of exactness. The figures above display two time series plotted on the same X and Y-axis. As can bee seen, they are not exact matches but the r-squared value of the Pearson correlation between the series is 0.99. This is almost an exact match, and suffices as an excellent correlation. If the sample set is large, two lists may correlate highly even if they contain outliers. This is possible due to the mean staying close enough to the median, of the time series, to keep the correlation close to one. This is a benefit of correlation, where values do not need to be exactly alike to have likeness.

Spearman Correlation: Simple Non-Parametric Cluster Recognition

The Pearson correlation coefficient is a parametric ranking statistic. As mentioned earlier, being parametric requires that the data come from a known probability distribution. Another assumption of the Pearson correlation coefficient is that the data come from the same domain, and are of the same type. This means that the data to be compared must be of the same units or category. These assumptions are often abused, and result in untrustworthy results.

Non-parametric ranking measures exist to allow likeness comparisons between cross-domain and distribution less data. Often the distance measures are similar to likeness measures. A list of ranks, R_x, is defined as a list of the ranks for a corresponding set of raw values. The rank may be defined in many ways, and a simple numerical ranking is a typical route. Ranking becomes complicated when ties exist. Ties can be defined by

$r_{x_2} = r_{x_{12}}$. A through understanding of the domain from which the data points are sampled from is essential to choosing the correct ranking algorithm. Many algorithms have been developed to handle ties within data lists, and a few are reviewed.

Simple ties equivalent rank is a simple, yet intuitive ranking algorithm. The following steps, with an example, define the algorithm.

1. Sort list of values by numerical order, either lowest as highest rank or highest as highest rank.
2. Ties are handled by assigning an equivalent rank to each tie.

The following example is a sample where highest value is assigned the highest rank. For a set of raw values V, R represents the corresponding ranks in appropriate position.

Example of simple ties equivalent rank

$V = \{11,12,13,13,14,15\}$
$R = \{1,2,3,3,4,5\}$

Ties shift rank is another simple numerical ranking algorithm. Again the following steps, along with an example, define the algorithm.

1. Sort list of values by numerical order, either lowest assigned highest rank or highest assigned highest rank.

2. Ties are assigned an equivalent rank, where the value following a tie is assigned a rank equal to the rank of the tie plus the number of matches within the tie.

The following is an example where highest value is assigned the highest rank. Again V represents the raw value set and the set R contains the appropriate rankings.

Example of ties shift rank

$V = \{11,12,13,13,14,15\}$
$R = \{1,2,3,3,5,6\}$

Disregard ties rank, is an algorithm that basically ignores ties altogether. This algorithm is defined as follows:

1. Sort list of values by numerical order, either lowest assigned highest rank or highest assigned highest rank.
2. Ignore ties, assigning ranks in order once the sort has been made.

Again the example below illustrates the disregard ties rank algorithm. The set V contains raw values and R is the corresponding set of ranks.

Example of disregard ties rank

$V = \{11,12,13,13,14,15\}$
$R = \{1,2,3,4,5,6\}$

$$SRO = 1 - \frac{6\sum_{i=1}^{n}\left(rank(X_i) - rank(Y_i)\right)^2}{n(n^2-1)} \qquad (3.21)$$

Spearman rank order, SRO, correlation has the advantage of being non-parametric, along with lacking any restraints on the domain of the data. Correlation along with partitioning is an elementary method of clustering. Performing a cross correlation, regardless of correlation type, upon a set of time series can identify sets with some commonality. This likeness exists due to high correlation, or high inverse correlation between time series. This allows grouping of similar series into a cluster with similar characteristics. For example, having a set of time series represented as a matrix, *M*, where:

$$M = \begin{bmatrix} t_{1,1} & .. & .. & t_{1,j} \\ .. & .. & .. & .. \\ .. & .. & .. & .. \\ t_{i,1} & .. & .. & t_{i,j} \end{bmatrix}$$

A typical cross correlation matrix, C, is of the form,

$$C = \begin{bmatrix} corr(t_i,t_i) & corr(t_i,t_j) \\ corr(t_j,t_i) & corr(t_j,t_j) \end{bmatrix}$$

Where $corr(t_x,t_y)$ represent the cross correlation value for a comparison of time series X and Y. It must be noted that if $corr(t_x,t_y) = 1$ then $corr(t_x,t_y) = corr(t_y,t_x)$. By viewing only rows or columns, not a mixture, looking for highly correlated pairs allows a researcher to spot clusters of numerically like pairs.

Introduction to Learning Techniques

Learning is the basic process of gaining knowledge from data. Learning techniques can be classified by being supervised or unsupervised. Supervised techniques, quite simply, require intervention from the teacher during the learning process. This intervention allows the learning technique to correct its mistakes by using input from the teacher. To accomplish this training of the model, the assumption of independence between inputs must be made. The statistical models discussed within this document assume independence, and therefore are unsaturated. Saturation refers to the inclusion of normal factors along with factors that represent dependencies between input variables. Building a model with this setup will reduce the reliability of the model to make consistently accurate predictions. As stated previously assuming independence is typical of statistical model construction that takes the form:

$$y = \beta_0 + \beta_1 x_0 + \beta_2 x_1 \ldots + \beta_{n+1} x_n \qquad (3.22)$$

Where β_0 represents the intercept of the x-axis, and β_{n+1} represents the weight of the corresponding independent input x_n. Supervised techniques are classified as so due to the 'training set' that is used to construct the model. Supervised model construction involves feeding data to a learning machine while using the errors between the expected output (original value), O_e, and the actual output, O_a. These errors, or residuals, are used to

tighten the accuracy of the model for the entirety of the training set. Some examples of supervised learning techniques are statistical regression, and artificial neural networks.

Unsupervised Learning Techniques

Unsupervised learning techniques are those that analyze the data without previous knowledge of patterns or trends that do not require a training data set. Unsupervised techniques are typically hands-free algorithms, meaning all of the results must be scrutinized after the machine has interpreted the data. This is unlike supervised learning where, if unhappy with the results or the size of the residuals, the researcher may either retrain on a different training set or use the same data and attempt to gain further knowledge prior to prediction by retraining the machine on more data. In doing so, this leaves the prediction data window smaller, due to an increased size of training data. An important note when using unsupervised learning is to expose the system to the majority of features during training. If new features arrive during testing or production use, the residual error will increase and can be high enough to make future predictions unreliable. Some examples of unsupervised learning are artificial neural networks, and Expert Systems. From the examples above you'll see that artificial neural networks fall into both categories, and this is so due to the learning algorithm implemented within the network.

Artificial Neural Networks

As noted artificial neural networks are both supervised and unsupervised. Table 3.3 lists the paradigms of artificial neural networks.

Table 3.3 – Paradigms of Artificial Neural Networks

Unsupervised	**Supervised**
Hebbian Learning	Error-Correction Learning
Competitive Learning	Stochastic Learning
	Reinforcement Learning

Some of the benefits of unsupervised learning techniques over their supervised counterparts are:

1. The ability to adapt itself to non-linear problems, for which basic statistical techniques are too weak to predict accurately.
2. ANNs have the ability to learn and control its outputs in real-time, by adjusting weights of the outputs of error functions.
3. Uniformity, where it has the ability to ignore restrictions of contemporary modeling techniques such as distribution dependencies.

Unsupervised learning has many restrictions, which limits their use mainly to non-linear problems, some of which are:

1. Computationally expensive in comparison to contemporary modeling techniques. ANNs require a lot of iteration of error calculations to achieve a good fitting model.

2. Unsupervised learning techniques are of black-box design, due to architecture. The limited configuration is in the node weighting and choice of error function. A design of this type requires that the researcher trust the network.

The goal of supervised learning is to guess the weight matrix so that the network can minimize the resulting residual error. Supervised ANNs require that training be performed prior to prediction. Figure 3.6 illustrates a simple neuron, or node, setup.

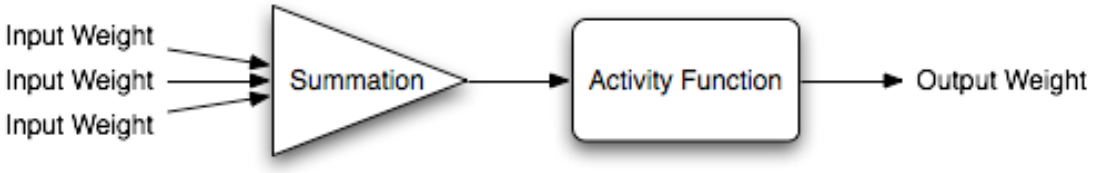

Figure 3.6 – Simple Neuron, Component of Artificial Neural Network

Neural network classifications include feedforward and recurrent. Within each of these there includes subclasses. Figure 3.7 illustrates a simple feedforward network with one layer of hidden nodes, which perform the activity calculation.

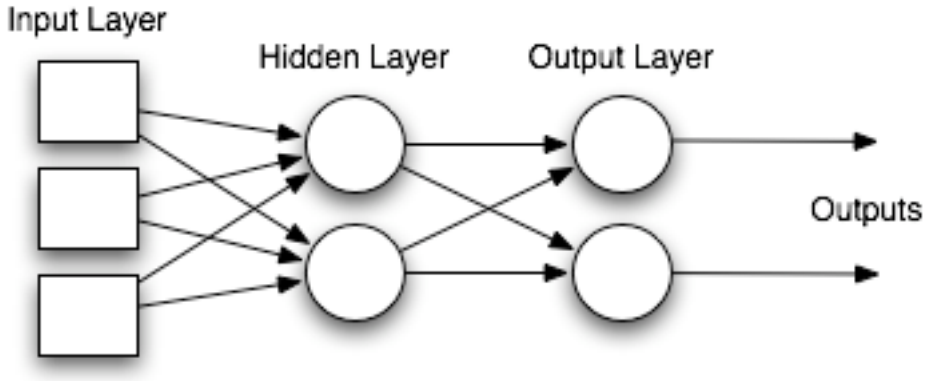

Figure 3.7 – Simple Feedforward Neural Network

An adaptation to the simple feed forward involves looping within the network, which are represented by recurrent networks. Figure 3.8 illustrates an example adapted from figure 3.7 to represent a typical recurrent network.

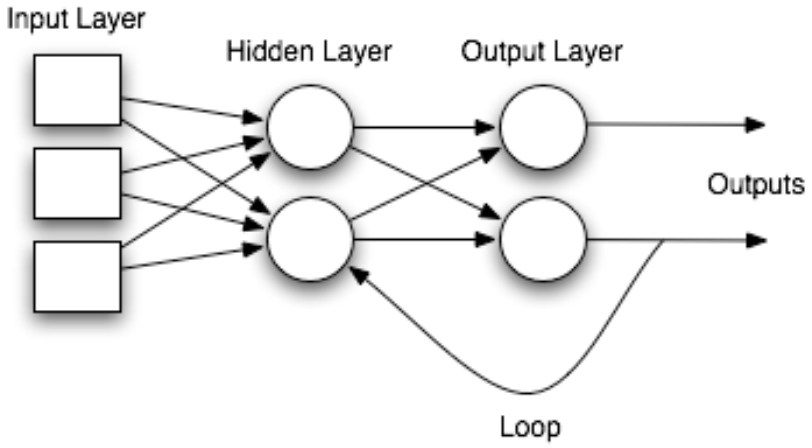

Figure 3.8 – Example of a Recurrent Network

There are multiple classifications for neural networks but they all perform the same basic operation. This operation is to minimize an output error function. In a supervised error-correction network calculating residuals from known and network-calculated outputs

does minimization. One of the most well known error-correction algorithms is known as backward propagation. This involves correcting an error function in two steps. The first step in a traditional weight calculation is the use of the activity functions. The second is a backward propagation of the error correction being applied to the originally calculated weights. The error correction function for backward propagation is typically the average squared error energy, which is calculated by

$$E_{av} = \sum_{1}^{n} n * \left(\frac{1}{2} * \sum_{1}^{j} (d_j(n) - y_j(n)) \right) \qquad (3.23)$$

Where d_j = trained output, and y_j = calculated output. For more information, and a full derivation, on the error function and weight deltas please refer to Kantardzic [12]. After multiple backward propagations, the error function is intended to converge. This convergence allows for the best possible, i.e. lowest, error across the network. At this point the network is ready for testing of its predictive capability.

Supervised Learning Techniques

This classification of learning techniques involves human interaction. They require the researcher feed the learning algorithms data, and perform interpretation of the results. Of the known supervised learning techniques, statistical regression is the most popular method for predictive analysis and k-means clustering for clustering analysis. This type of learning is the easiest to use, and therefore are a good starting place for attempting to apply learning to a dataset.

Linear Regression

To predict based in a linear combination of other values, a researcher has linear regression as the tool of choice. Regression stems from the problem of determining the sample mean of a joint distribution, where the mean is expressed as $\mu_{y|x}$. Formally linear regression can be expressed as $\mu_{y|x} = \beta_0 + \beta_1 x$. The basic expression for linear regression, expressed in terms of a dependent variable y, is $y = \beta_0 + \beta_1 x$. There are two popular forms of basic linear regression, linear and logistic regression. These two are separated by their classification of input variables. Logistic regression is used to predict the logarithmic value of a dependent variable, using probabilities as independent variables. Logistic regression is represented formally as $\log(y) = \beta_0 + \beta_1 \Pr_x$. These probabilities are of categorical variables. Linear regression, as stated previously uses numerical independent values as inputs to predict a numerical value as an output. In these simple forms regression is easily understood, but in practice typically there are many independent inputs. This multivariate designation is referred to as multivariate linear regression. To solve multivariate linear regression, the method of Least Squares is applied. The method of least squares hinges on the fact that for a linear fit, a straight line centered among the data is the best fit for prediction.

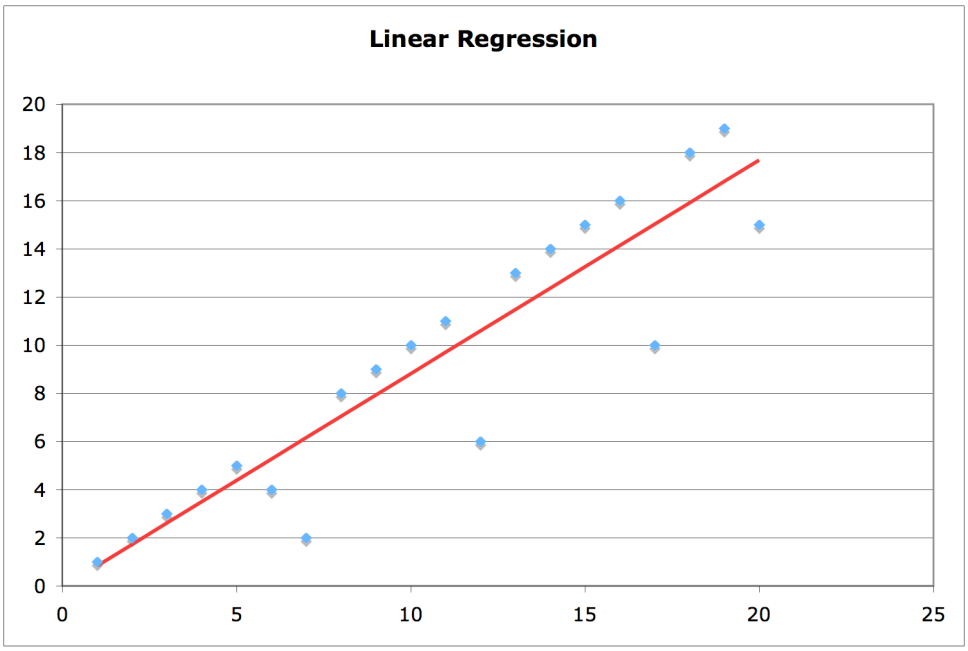

Figure 3.9 – Example of a Linear Regression Fit

Minimizing Δy by achieving the lowest average error, from figure 3.9, is the premise for least squares approximation. To solve the regression equation,

$$y = \beta_0 + \beta_1 x,$$

Where β_0 is the x-intercept and

$$\beta_1 = \frac{SS_{xy}}{SS_{xx}}. \qquad (3.24)$$

where, from the previous equation, values SS_{xx} and SS_{xy} are the sum of square values for x and y respectively. Multivariate regression requires that the methodology of least squares be extended to more than one input variable. To accomplish this the researcher

uses the matrix notation of least squares defined from the previous equation $y = \beta_0 + \beta_1 x$, and more generally $y = \beta x$. Originally x was a list of data, and this can be viewed as a vector. To expand on this, using multiple lists, i.e. multiple independent inputs.

$$x = \begin{matrix} d_{1_0} & .. & .. & d_{n_0} \\ d_{1_1} & .. & .. & .. \\ .. & .. & .. & .. \\ d_{1_n} & .. & .. & d_{n_n} \end{matrix}$$

And to solve for y, and expose the weights β of $y = \beta x$, a researcher uses the matrix form below.

$$\beta = (xx^t)^{-1}(x^t y) \qquad (3.25)$$

where the factors (xx^t) and $(x^t y)$ square the dependent and independent matrices. This methodology can be used assuming the matrix x has an inverse i.e. is non-singular.

Logistic regression may also be used by researches to predict variables. More specifically it is used to predict the probability that an event will occur based on the input probabilities. Logistic regression can be expressed formally as,

$$\log(\frac{\Pr(x)}{1-\Pr(x)}) = \beta_0 + \beta_1 x \qquad (3.26)$$

Again statistical regression is one of the most popular supervised learning techniques. Although ANNs can produce a similar output for line fitting, regression is far more practical for its one-pass use. As noted, when using ANNs, the typical error calculation is a back propagation that involves a recursive procedure that costs more than a simple one-time least squares calculation.

Clustering Techniques

Clustering is also another popular supervised learning technique, and is of importance within this thesis. The human eye can easily perform clustering in less than three dimensions. As space size increases it becomes numerically complex and virtually impossible for the human to recognize existing relationships. Clustering can be best defined as numerical (logical or ordinal) relationships that exist between objects (and dimensions). These numerical relationships are usually defined by the minimization of an error function, for instance mean square error. Mean square error, or MSE, is defined as

$$\sum (x - mean_c)^2 \qquad (3.27)$$

Where x = data value within cluster, and m = the cluster mean. The author is aware of the ability to heavily skew this calculation with outlier values. From this the author recommends the use of median square error, which is defined by

$$\sum (x - median_c)^2 \qquad (3.28)$$

Clustering is a very simple methodology, and the $(x - mean_c)$ within the MSE can be replaced by many other distance functions. Table 3.4 lists other distance functions that are popular.

Table 3.4 – Distance Measures

Distance Measure	Formula
Euclidean	$\left(\left(\sum x - mean_c\right)^2\right)^{\frac{1}{2}}$
City-Block	$\sum \lvert x - mean_c \rvert$
Minkowski	$\left(\left(\sum x - mean_c\right)^p\right)^{\frac{1}{p}}$

For more distance measures please refer to Statsoft and Kantardzic [9,12]. The most common clustering technique is the k-means clustering method. It has been used in research that is contained within this document and is defined as

1. Select an initial set of clusters.
2. Assign each value to the closet cluster center.
3. Compute the new cluster centers, or centroids.
4. Go to steps 2 and 3 until the cross cluster error converges.

The cross cluster errors is defined as

$$\sum MSE_c \qquad (3.29)$$

Once the cross cluster error converges, or reaches a user-defined limit, the clustering process terminates. As expected this process can be computationally expensive, and software packages allow you to limit the number of iterations of steps two and three. The clustering that takes places within this document has limited iterations to lower total analysis time.

Clustering's tough challenge is to determine the number of distinct features, i.e. the number of clusters. To do so may require domain knowledge or, conventionally the use of Principal Components analysis. Principal Components analysis (PCA) is loosely defined as finding the features in a multi-dimensional space that account for a large portion of the data set's variance. More technically PCA is defined as a process, not just an equation. It is a transformation process on original data allowing exposure of high variance components. The following steps accomplish the process:

1. Calculate the mean deviance residuals, R.

   Where $R_i = x_i - \bar{x}$ and $x_i \in X$ (3.30)

2. Calculate the covariance matrix, C, of mean deviance residuals.

   $C = \frac{1}{n-1} R * R^T$ real values only (3.31)

3. Find eigenvectors, EV, and eigenvalues, E, for the covariance matrix.

   $C * E = E * EV$ (3.32)

4. Sort eigenvalues in decreasing order based of magnitude.

5. Choose the most significant eigenvalues to be used as a representation of the variance associated with the data set.

The PCA is a key process to use in determining unique features in the data set. PCA's functionality derives from its ability to find linear projections of the variance of the data set. The most distinct of these projections is a useful feature within the data. PCA is used in this document to identify the number of clusters that should be assumed prior to the clustering process.

POWER GRID DATA ANALYSIS

Characteristics of the System

An explanation of the data set's history is essential to obtain a full understanding of the goal of the research and the complications that were discovered during the research. The US Environmental Protection Agency, established on December 2, 1970, is charged with the job of drafting and enforcing environmental laws enacted by the Congress of the United States. A subsection of this enforcement includes collecting data from power plants and monitoring the levels of environmental pollutants emitted during the generation of electricity. The data used in the analysis are in the public domain, and have been for many years, which allow access to historic data. The data contain many factors that affect our environment, and the power industry. They include things such as the level of oxides of nitrogen, fuel type, and generation in megawatt-hour per generator, etc. Many factors are measured by, or are reported to the EPA. This research is only interested in mining a small subset of the data, specifically generation unit average production (generation) per hour (MWH). Although this generation is reported hourly on a generating unit by generating unit basis, these values can be summed to obtain the desired plant-wide generation. The generation is compared to the maximum theoretical output, or capacity, of the generating unit. Plant capacity is calculated by summing each generating unit's individual capacity. A generating unit's capacity is defined as the maximum electrical energy that can be generated. For those readers not familiar with power plant design, a brief explanation is in order. There are two types of energy, kinetic and potential. Generation fuels contain potential energy and those include coal, natural

gas, uranium, bio-fuels, etc. To extract this potential energy the process uses a boiler. A boiler is lined with metal tubes, and these tubes are filled with water. This boiler transfers the potential energy from the fuel by creating a large repository of heat from burning fuel. The kinetic energy, in the form of heat, in the flame front is transferred into water, via conduction through the boiler tubes, which is then pressurized and heated resulting in steam. Kinetic energy in the form of pressurized steam turns the turbine shaft, which is connected directly to the generator. The rotation of the generator shaft causes repetitious generation and degradation of the magnetic field that converts this rotational, although kinetic, energy into potential energy. This potential energy is in the form of voltage due to induction of a current into a winding within the generator after the magnetic field has degraded completely. The output of the generator is a unit of energy referred to as a watt. Typically the unit used in the Power Industry is the megawatt, which is 1,000,000 watts, where a watt is defined as:

$$watt = \frac{joule}{\sec ond} = volt * ampere \qquad (4.1)$$

Goal

This document, and the research behind its construction, is based on analyzing data looking for patterns. The goal of this research is to discover characteristics within electricity generation between power plants who use natural gas as their primary fuel so that they may be applied to modeling system values. If known clusters of generating units exist, a researcher may be able to minimize the inputs to a global statistical model with this information. This is driven by known factors that effect the generation of gas plants.

Existing contractual stipulations, an example of a physical characteristics that dictates plant operation, should be apparent after characteristic discovery. Although this goal is narrow in the sense of the power industry, it can be broadened to encompass all power plants globally. Solving a problem of this magnitude is not feasible for this study. With these characteristics, a miner may generate some association rules that result in semi-predictable operational characteristics or even the ability to reliably predict output via a generation model. Although this research is performed solely for educational purposes, it has a direct impact within the power industry. This impact leaves the author to believe that a nationwide mining process would likely yield fruitful results and would be an exceptional addition to this document.

Type of Data

The Environmental Protection Agency monitors many factors including emissions of power plants. Many of these factors are focused solely on emissions, but the generation data are also included in the historic data [1,2]. The data for this research is extracted from the Environmental Protection Agency's Electronic Data Registry [1]. The data used within this document are primarily generating unit output data of natural gas-fired power plants, which are recorded hourly. EPA records show that there are over 1700 gas-fired generating units nationwide. From these facts it can be concluded that the dataset size is greater than 14,892,200 hourly generation values. Boundaries for the generation values are zero up to plant capacity. This data are typically hard to pre-aggregate, unless it has a direct implementation. A miner maybe interested in daily, hourly, monthly, or even yearly generation values. There is also a possibility to aggregate the values with respect

to NERC region, state, or zip code. As you can see it would be hard to pre-aggregate the raw data. Although the data are cumbersome, once stored in a data warehouse, they become easier to manipulate into usable aggregates. The research for which this document exists, is concerned with granularity at the hourly level. Therefore all statistical analysis will be done on an hourly basis, and all data stored are hourly in raw form. Another caveat of this data set is the existence of nulls. Nulls are typically in place due to the lack of the correct data, and this is the case for the data in the EPA's Electronic Data Registry. Although the null data can be ignored, or even cast to zero, it must be recognized that missing data are different from data that have a magnitude of zero. Therefore casting to zero is a bad idea, leading to false relationships as a result of mining. Specifically the data comes from forms submitted to the EPA, as mentioned previously. The forms defined by a reference guide [4] allow the researcher to extrapolate the information from its cryptic form. Once extrapolated, the data can be examined.

The data used in this analysis are not the only publicly available data for power plant electricity generation. The EPA, EIA, and NRC organizations supply the market with generation information. The EPA dataset used, from the EPA's Acid Rain Program, are the data recorded in compliance with Title IV of the Clean Air Act of 1990 [5]. The Acid Rain Program was instituted to allow for continuous monitoring and reporting of power plants. A limitation was placed upon the monitoring to include only generating units of capacity greater than 25 MWh. There is a plethora of emissions reporting that is done by plants to comply with the act, but this document is only concerned with the generation data. The data reported in compliance with the Clean Air Act are on an hourly basis; this

is an important fact when comparing data sources. The Energy Information Administration, or the EIA, is another branch of the Government that monitors power generation. The EIA only requires plant operators to report monthly. The generation data must be reported if the generating unit is connected to the electric power grid and has a capacity greater than 1 MWH [6]. The agencies previously mentioned require reporting for units regardless of fuel type. The Nuclear Regulatory Committee, NRC, regulates generation from nuclear powered units only. Due to the hazardous nature of nuclear fuel, the regulations for its use must be tighter. Due to the granularity of the sampling, the EPA dataset has been chosen for analysis. Although the EIA dataset contain more generation values, due to containing more units, the smaller generating units are of little concern.

DATA COLLECTION AND PREPROCESSING

Introduction

As mentioned earlier the analysis has been performed on hourly average generation values. The EPA collects these values from SCADA systems that reside in the Power Plants. This data are sent to the EPA in an encoded format that must be extracted via the explode.exe [3] program offered by the EPA as a downloadable tool. The program is a Disk Operating System (DOS) utility that must be ran within Microsoft Windows. The data used in this analysis was not in the raw form and must be decoded. Once decoded the data can be stored into a relational database and processed further. The output of the software is an ASCII file that is readable. But from this raw ASCII file, there must be further processing to obtain the data. The explanation of the format for the raw ASCII file is located in the EPA EDR reference [4], and is left as an exercise for the reader. Scripts, not written by the author, were run to remove, from ASCII files, and insert the data into appropriate tables. The data are published quarterly and require that the parsing scripts be ran quarterly to accumulate historical data for analysis. Although the data are now in tables and can be accessed via SQL, the data are still coded in the format required by the EPA. Aggregate tables are then created to allow a researcher, who has a grasp of the domain, the ability to retrieve the historical data with the EPA reference as a guide. The author took it upon himself to create these aggregate tables for the data that is analyzed within this thesis. As said previously, the data examined within this document are generation data. There is a large amount of data that comes from the EPA, and this research is focused on a small percentage of that data. Therefore the database that

contains the data is small and the structure is more scrutinized allowing for quicker access to data. To fill these tables, the author retrieved a subset of this larger EPA data set from another database.

Discussion of Observations

Overall the data itself are somewhat sparse. An example of the sparseness is a missing hour for a unit's average hourly generation. This is typical of the data problems seen within this data set. To overcome this, the researcher accepted the data as missing and flagged, as "don't cares". To forecast this missing value, could result in serious error. The data used represents generation, which is very dynamic and is driven by supply and price. If supply decreases and the current market price of the energy is higher than the costs of generation, the generating unit output is increased if possible. These decisions are not made at the plant level and usually come from a corporate load center. Although the aforementioned councils oversee the generation, for the most part, the corporations run the units primarily for profit. With this in mind one can understand the dynamic operation of the units, during normal operation. To predict the output of a generating unit is possible with high degree of accuracy if the unit is stable. Specifically, within this thesis, we are concerned with patterns that are connected with change. Therefore the missing values, representing hourly average generation, were flagged as "don't cares" during the coding procedure. Flagging the missing values as so, requires that the researcher must not ignore their significance.

Generation, as said previously, must stay within the constraints of capacity. Another skewing element, of generation data, is false reporting. This may include inflated or deflated generation values, due to malfunctioning equipment that is responsible for reporting. To accommodate this a researcher may handle this is many ways and one way to handle this is to remove the outlier generation values. Obviously if the generation is above capacity this is an easy task. But what happens if the reporting unit is malfunctioning and the unit is operating below capacity? The generation values are wrong and hard to identify as incorrect. To compensate, the researcher must only disregard the true outliers. Malfunctioning equipment is reporting error and cannot be controlled. The data analyzed in this document was cleaned using the methodology described above of removing obvious outliers. Obviously the missing values allow for uneven lengths of data sets. This is corrected by using a flag as a placeholder for the missing timestamp's value. Again this is accommodated in the data set that has been analyzed for the research in this document. Another way to repair skewed generation values are to replace the missing value with the generation from the previous day's hour. This methodology works well if the unit is stable. Stability is loosely defined as a non-peaking unit. Peaking units, although not clearly defined, are those that do not obey cyclic patterns. Nor do these units operate monthly in a band of generation that equates to a low sample standard deviation of generation values. Most generating units have cyclic generation output patterns. This cyclic pattern is defined by 'on-peak' and 'off-peak' hours. Although these hours are not defined clearly, one can estimate these empirically. To estimate these hours, a researcher must average the daily minimums for weekday data. Weekend data must have a separate model due to different load requirements. Basically

load changes around 07:00 EST, increasing to a level that is appropriate for business operation. Again the load changes around 22:00 EST, decreasing to accommodate home usage. The cyclic pattern is explained in this manner. This argument solidifies the solution to replace with the previous day's hour. Moving farther back in time for generation replacement, like the same hour from the last week, could result in error due to load changes. Load changes are a result of temperature, seasonality, population density, etc. There are many ways to approximate missing values and all will result in some error. Due to this assurance, the researcher chose to keep the missing values as missing. All patterns found with large numbers of missing values will be suspect.

Data Population

To populate the data into a separate database, aside from the warehouse, the researcher must create SQL to retrieve and move the data. Movement of the data was easy, requiring the use of importing software. The DBMS used for the research was PostgreSQL. The software used to import the data, database-populator, is provided by numericaljava.org. This software will allow a user to import data, in most any form. Prior to loading data, the researcher contemplated on a way to repair the data while still in a manageable form (a relational database). To accomplish this repair of missing values, using the flagging methodology, one can use advanced SQL to complete the repair and allow a researcher to load the data intact and repaired, ready for analysis. To repair the data with SQL, the use of packages created by someone other than the author were used. This package allows a query to create a resultset of appropriate timestamps. These timestamps represent all valid timestamps over the range of data. For the learning data used in this research, the

range was all hourly values from April of 2002 till April of 2003. The researcher was not able to obtain data for the full year of 2002. The data for that year was not complete, and offered many more errors that would require estimation and repair prior to use. The choice of a staggered year was best for the analysis. Once the timestamps are correct, using the left outer join operation in SQL will allow the researcher the ability to create values where there are no values. The left outer join operation is used to associate one column to another based upon the identifier from the first column. If table A contains a list of timestamps that are complete, and table B contains a list of time stamped data, the left outer join operator will connect data with its appropriate timestamp. If there are missing data for specific timestamps, in table A, the operator will return a NULL value for this data form table B. This is extremely useful in the creation of missing data. Also to accomplish the flagging of missing values, the researcher may use the NVL function (Oracle specific) or COALESCE (general SQL) to cast NULL values to *any* desired value. Once a result set is obtained from the creation SQL, the data may be loaded via the database-populator software.

Discussion of Dimensionality

The data, analyzed by this document, are from the Environmental Protection Agency's Acid Rain/OTC Hourly Emission Data. The data represent expelled emissions by generating units who burn natural gas as the primary fuel. As of 2005 there were 1439 power plants, 849 of which burn pipeline natural gas that report to the EPA in EDR format. Each plant may contain multiple units. In total there are 4752 generating units that report hourly emissions, and of that there exist 2437 pipeline natural gas-fired units.

For a single unit there are 8760 hourly reports made for a year (non-leap). This equates to 21,348,120 hourly generation values that must be processed. Each unit level set of hourly values must be coded. This requires computing basic statistics, as mentioned previously, on every unit for the time period analyzed. For instance to compute a simple Pearson cross correlation matrix, a researcher must instruct a computer to make $\frac{n^2 - n}{2}$ calls to the Pearson correlation calculation. This is fully optimized avoiding diagonal and lower triangular calls, which may be performed once prior to calling the Pearson correlation calculation method. For the data analyzed, that would equate to 2,968,266 calls to the correlation calculation method. Along with the high number of calls to the method, the ability to view these results, of this dimension, are questionable. Most mining and data analysis is accompanied by a GUI representation of the data allowing the researcher to confirm the results. This visual representation of the data is truly essential, and for a data set with high dimensionality introduces complications. To effectively view and store correlation, clustering, and motif-discovery results from a data set this large there must be database integration. Using a database to store results allow easy manageability but require special software to load and display the data. To avoid these issues, the author chose to use the aforementioned data set, while narrowing the scope of the analysis. Specifically the analysis was performed on the first 50 generating units of the overall set. The idea that associations can be made from successful discovery in a subset should imply that in a larger subset the same associations would hold true. When moving to the full set, the restrictions placed on pattern discovery may be tightened resulting in more dependable relationships. This is due to the requirement of a higher number of matching patterns.

Does the data exhibit normality?

After the data has been loaded, and is full with regards to timestamps, the data are ready to be analyzed. Generation data are very sporadic and diverse, as previously mentioned. The diversity of the data set makes it almost impossible to compare the output of two generating units. To mend this diversity, a domain expert, chooses to normalize the data with regards to unit capacity. This is a logical step, as the unit cannot operate above unit capacity. There are many other methods to achieve normalization, and a typical example is to assume a normal distribution [16]. The reference refers to time series as very often originating from a normal distribution. The author's domain experience can confirm how this property does not hold to energy data. Since the analysis is performed on energy data, and since the energy domain is a fruitful field of study, this classification cannot be universally applied. Below is an analysis concerned with disproving this assumption. Initially one must understand the state of the available data. Please refer to Table 5.1 for a summary of basic statistics for the natural gas units analyzed for the calendar year of 4/2002 through 3/2003. The included statistics are the minimum average-hourly generation, maximum average-hourly generation, sample standard deviation of average-hourly generation, arithmetic mean of average-hourly generation, and the count of average-hourly generation values that are not null (otherwise included in the other calculations). This table is included in the appendices to allow the reader to further examine the sparsity of the data. The generating unit chosen was unit 6A at the James M. Barry Generating Station located in Bucks, Alabama connected to the Mobile River [36]. This unit was chosen due to its high non-null value count and high average generation

values. A large standard deviation, as this unit posse, of generation is an indicator of a flattened normal distribution (which leads to a uniform distribution).

Hypothesis:

H_o: Generation data isn't sampled from a Normal Distribution

H_1: Generation data is sampled from a Normal Distribution

The author chose the Chi-Square test statistic, X^2 and its "Goodness of Fit" test to validate the hypothesis. The Goodness of Fit, using the Chi-Square test statistic is defined as

$$X_0^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \qquad (5.1)$$

Where H_0 is accepted when

$X_0^2 < X_{\alpha,df}^2$

df = Number of degrees of freedom

α = Tolerance

k = Number of bins

Table 5.1 contains the results from the test and some basic statistics for the unit tested.

Table 5.1 - Goodness of Fit

Data Set Statistics	
n	8759
Arithmetic Mean	152.15
Sample Variance	8133.68
Sample Standard Deviation	90.19
Range Minimum	0
Range Maximum	312
Number of Bins	10
Bin Width	31.2

Bin Min	Bin Max	Pr	O_i	E_i	$\frac{(O_i - E_i)^2}{E_i}$
0.0	31.2	0.0900	2112	788.3100	2222.6728
31.2	62.4	0.0698	22	611.3782	568.1699
62.4	93.6	0.0983	40	861.0097	782.8680
93.6	124.8	0.1228	20	1075.6052	1035.9771
124.8	156.0	0.1983	15	1736.9097	1707.0392
156.0	187.2	0.0720	3124	630.6480	9857.8037
187.2	218.4	0.1175	2077	1029.1825	1066.7899
218.4	249.6	0.0913	587	799.6967	56.5713
249.6	280.8	0.0631	497	552.6929	5.6120
280.8	312.0	0.0769	265	673.5671	247.8255
				χ_0^2	17551.33

$$\alpha = 0.95$$

$$k = 10$$

$$p = 2$$

$$df = 10\text{-}2\text{+}1\text{=}9$$

$$\chi^2_{(1-\alpha,df)} = \chi^2_{(0.05,9)} = 23.59$$

$\chi^2_0 > \chi^2_{(0.05,9)}$ Therefore reject H_0 and accept H_1

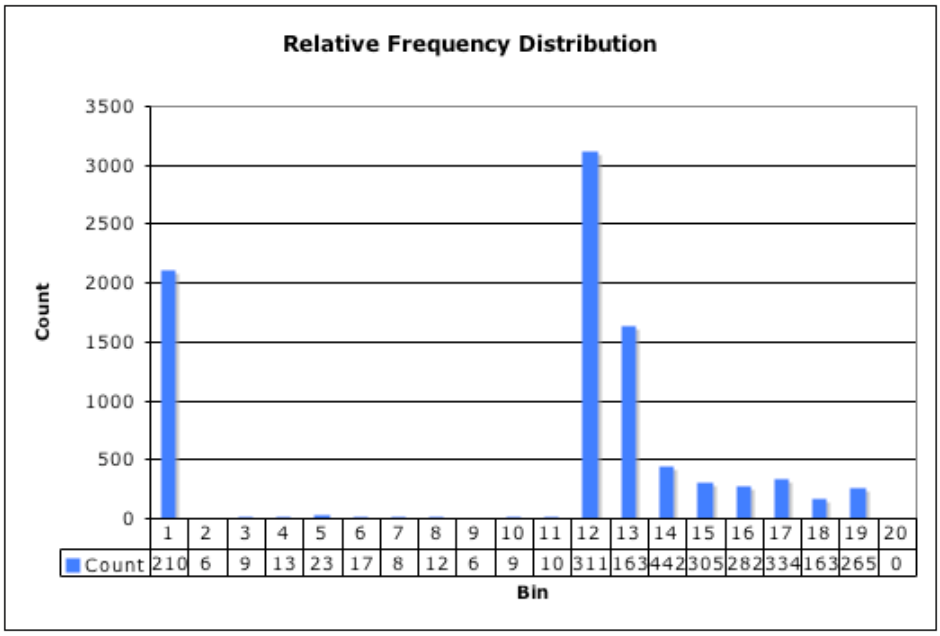

Figure 5.1 – Relative Frequency Distribution for Unit 6A at Barry Generating Station

The results of the test show that indeed this set of energy data, represented by a unit displaying typical operation, does not originate from a normal population. Viewing the relative frequency distribution was a good indicator that the sample was not of a normal population, but the test statistic proves it with certainty.

MODIFIED K-MOTIF DISCOVERY APPLIED TO THE POWER DOMAIN

Review of Theory

The focus of the research was to discover sequences of coded generation values. There are two steps to this process, coding and discovery. Before discussing implementation, a review of the theory is required. Many researchers have worked in the area of sequence discovery but Eamonn Keogh, of the University of California at Riverside, focused specifically on formalizing the problem of sequence discovery in the time domain. In [18] he talks about the issue of coding values, or as he calls it segmentation. His idea of segmentation is based upon j predicted constant values, hence segments, where each prediction is representative of multiple points.

$$y - \bar{y} = \frac{S_{xy}}{S_x^2}(x - \bar{x}) \qquad (6.1)$$

His schema of segmentation is based on normalized error between actual data and predicted data,

$$e_i = \frac{\sum_{m=1}^{j} d_m^2}{j} \qquad (6.2)$$

where d_m^2 represents a distance measure between a predicted value and the actual value. The segmentation is repeated until there is only one segment, in which all segments are merged by choosing a minimum error value. This idea, although valuable, wouldn't work well for power data due to the unpredictable sequence of values associated with the demand of power. Dr. Keogh has another paper [40] that also brings up the idea of

coding. In this document he explains that all time series are normally distributed, that is to say that the probability of values are based on a normal distribution. Dr. Keogh and his colleagues also introduced the idea of K-Motifs, or sequences with large counts within a subset of data, and "don't cares" which are values that represent noise and should be avoided [40,16]. Formally a K-Motif can be defined as a subsequence, S, which contains a significant instance count that does not intersect (or minimally) with other motifs. Both of these ideas are used to create a custom methodology to discover useful knowledge from the power generation data set. Along with coding, Dr. Keogh is focused on optimal discovery algorithms for locating useful K-Motifs. His team offers an optimal algorithm K-Motif discovery algorithm, deemed EMMA [40]. To avoid non-computational complexity the author developed a trivial brute-force implementation. As proven in the earlier section, the assumption of normality is false and will be ignored; therefore, a new methodology, USD, Uniform Sequence Discovery is proposed and applied. USD is based on the idea that, with no other knowledge, a uniform distribution must be assumed in data that originated from a demand-oriented situation (i.e. a market). USD can be defined as a process consisting of three steps.

The expected result from the analysis performed within this Master's thesis is a set of association rules for both unit-level and plant-level data sets. This rule list is used as a local, non-global, set of rules that can be applied to future data as it is processed. A rule can be used as a predictor of operation (in this case). For instance, if plant A has an operation that can be described by the pattern {0,3,2,1} with a support of 125, and with a

confidence of 0.75, plant F will also exhibit the same operational pattern. The rule statistics, support and confidence are defined by:

$$\sup port_{Plant\_A}(pattern) = Count(pattern)_{Plant\_A} \quad (6.3)$$

$$confidence(Plant\_F \mid Plant\_A) = \frac{\sup port_{Plant\_A}(Pattern) \cap \sup port_{Plant\_F}(Pattern)}{\sup port_{Plant\_A}(Pattern)} \quad (6.4)$$

The support value is nothing more than a count of instances of a given pattern within the set of patterns for plant A. A support exists for each pattern instance, for each plant (or unit). The confidence is a ratio. The numerator is a count of patterns that are in the intersection of the sets of patterns for plant F and plant A. The denominator is nothing more than the support for plant A for the sought pattern. The confidences can be related to conditional probability although not a true probability, as the probabilities nor distributions been calculated.

The Coding Process

The first step involved coding of the raw data using a schema based on a uniform probability. This coding schema takes into account the idea of "don't cares" as data that should be avoided during the coding process. As mentioned previously there are missing data points, within the power generation data set, and these are identified by negative one. These were avoided during the coding process, i.e. not coded, and avoided during discovery. Specifically these "don't cares" are tolerated to an level set by the user of the discovery software. For instance if the user will tolerate at most two "don't cares", the software will sequence match as long as this limit is not exceeded. Since the software

seeks all possible sequences (searches entire space defined by the alphabet), this limit is desirable. Ignoring all possible sequences containing "don't cares" would be foolish as they might give insight and allow predictability. As can be seen below, the number of bins is critical to the coding process. The analysis performed within this Master's thesis is based upon four bins with values. With this said, the alphabet contain the symbols {-1,0, 1, 2, 3}. As mentioned earlier, if one were to look at the coded data it would be obvious that another code exists. This other code is the negative one, -1, which represents a "don't care" and cannot be changed nor overlooked. This "don't care" is attended to in the discovery process. The number of bins was chosen arbitrarily, with no real logic. Below is the simple coding schema based on a uniform distribution.

$$bin\_deviation = \frac{local\_\max imum - local\_\min imum}{number\_of\_bins} \qquad (6.5)$$

$$code = \frac{actual\_generation}{bin\_deviation} \qquad (6.6)$$

The pseudocode that follows explains the coding process.

Algorithm: Coding Schema

```
1. for each unit in listOfUnits
2. do
3.     minimumUnitValue ← getMinimumValueForUnit(unit)
4.     maximumUnitValue ← getMaximumValueForUnit(unit)
5.     binDeviation ← (maximumUnitValue – minimumUnitValue) / numberOfBins
```

```
6.     codedValues []
7.     for each timestamp in listOfUnitTimestamps
8.     do
9.            actualValue ← getValueFromUnit(timestamp)
10.           codedValue ← -1
11.           if actualValue == maximumValue then
12.                  codedValue ← numberOfBins -1
13.           else if actualValue != codedValue then
14.                  codedValue ← floor(actualValue / binDeviation)
15.           end if
16.           codedValues[timestamp] ← codedValue
18.    end do
19. end do
```

K-Motif Discovery Algorithm

The second step was to perform a brute-force discovery of K-Motifs, based on a search of the entire alphabet of codes. This involved development of a custom software implementation to perform a full alphabet discovery based on restrictions set in place prior to execution. Many other researchers have spent time studying optimized ways to search a space of all patterns [17,18]. Although the author admits that the brute-force algorithm is the most expensive when judged by complexity, but it must be noted that every optimization possible while staying in an Object Oriented Programming Methodology were performed and allow for fast execution. The K-Motif discovery algorithm is a trivial pattern-matching algorithm, and this type of procedure seems simple but becomes interesting when observing the sequences found using the current coding

schema. Trivial pattern discovery, or K-Motif discovery, can be described as the process by which one searches for every possible pattern in a list of patterns. For example one should find three matches of the pattern {1,1,1} in the listing {1,1,1,1,1}. This shows that patterns may intersect, and therefore are not mutually exclusive. There is no explicit requirement that demands nor contradicts this condition; therefore, it was assumed valid. Implementation of software was a major piece of the analysis and work behind this document. The software is a simple recursive call to a trivial pattern matching routine. A pseudocode implementation is shown below.

Algorithm: Sequence Discovery

```
1. listOfCodes ← {-1, 0, 1, 2, 3}
2. maximumPatternLength ← input from user
3. minimumPatternLength ← input from user
4.
5. for each unit in listOfUnits
6. do
7.     listOfCodedValues ← getCodedValuesForUnit(unit)
8.     for code in listOfCodes
9.     do
10.          foundAPattern ← lookForPattern(listOfCodedValues, code)
11.          if foundAPattern then
12.                storePattern(code)
13.          end if
14.    end do
```

```
15. end do
16.
17. function
18. lookForPattern(dataset, pattern)
19.    newPattern ← appendNewCodeToPattern(pattern)
20.    if lengthOfPattern(newPattern) < maxPatternLength then
21.           lookForPattern(newPattern) //recursive call
22.    else
23.           if newPattern.existsIn(dataset) then
24.                  return true //found a valid pattern
25.           end if
26.    return false
27. end function
```

The procedure of searching all possible patterns in a given data set is very expensive. This intentional complexity allows the researcher the ability to search the entire pattern space. Once sets of patterns have been discovered for a specific unit, they may be crosschecked with other units, and rules may be created.

Rule Creation

The third and final step is rule creation. After all of the counts of matching patterns between units have been accumulated, the researcher can determine useful K-Motifs. This set of matching patterns is extensive and those deemed useful exhibit high support, and confidences. Since the problem was to create rules based on relationships between

generating units and power plants based on operating characteristics, a typical rule will take the form of:

G1: Generating Unit 1

S1: Support for Unit 1

S2: Support for Unit 2

M: Sample Motif, for instance {1,1,1,2}

CONF: Confidence Value for G2 with G1 present.

Rule: If M and G1 with S1, then G2 with a confidence of CONF.

Explanation of Rule:

If the sequence M appears at generating unit 1 then the sequence will also appear at generating unit 2 with a confidence of CONF.

Example of Coding, and Motif Discovery

The entire purpose of using the proposed algorithm was to identify exact sequence matches across multiple time series. This is done via coding the data and brute-force discovery. Simple put, matching sequences are those that are included in the intersection between the two sets of sequences available between two time series. The may be many instances of the motif in both sets, and therefore is deemed a K-Motif.

Figure 6.1 – Sequence Discovery over Sets

Motifs are nothing more than temporal sequences. Using raw, decoded, data makes it challenging to discriminate using distance measures. Coding does nothing more than normalization of data. This is crucial to comparing time series when amplitudes of raw data vary between series. Below is a complete example between two time series. It includes representation of raw data, normalization, sequence discovery, and rule presentation.

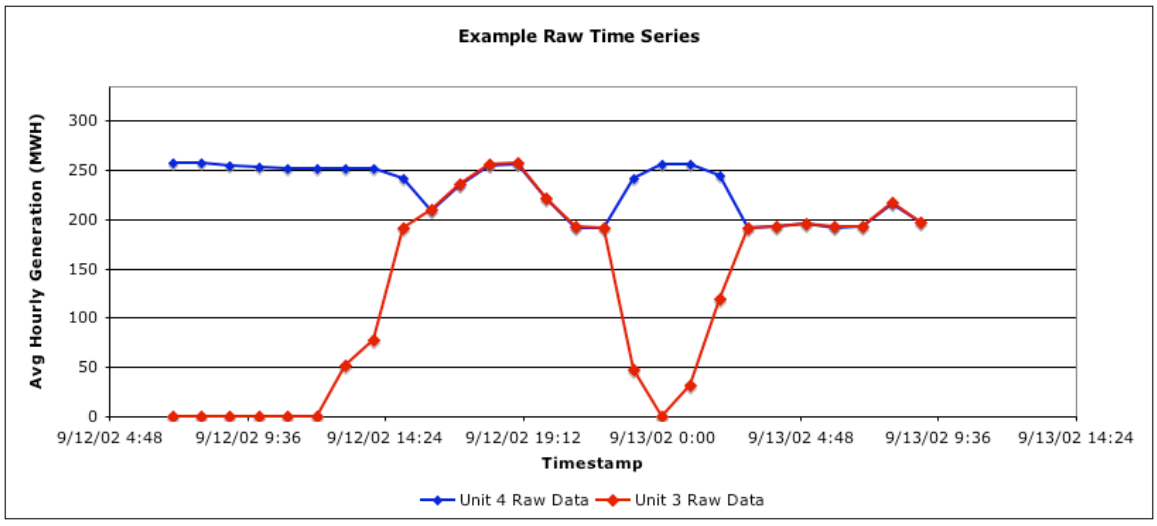

Figure 6.2 – Example of Two Time Series in Raw Form

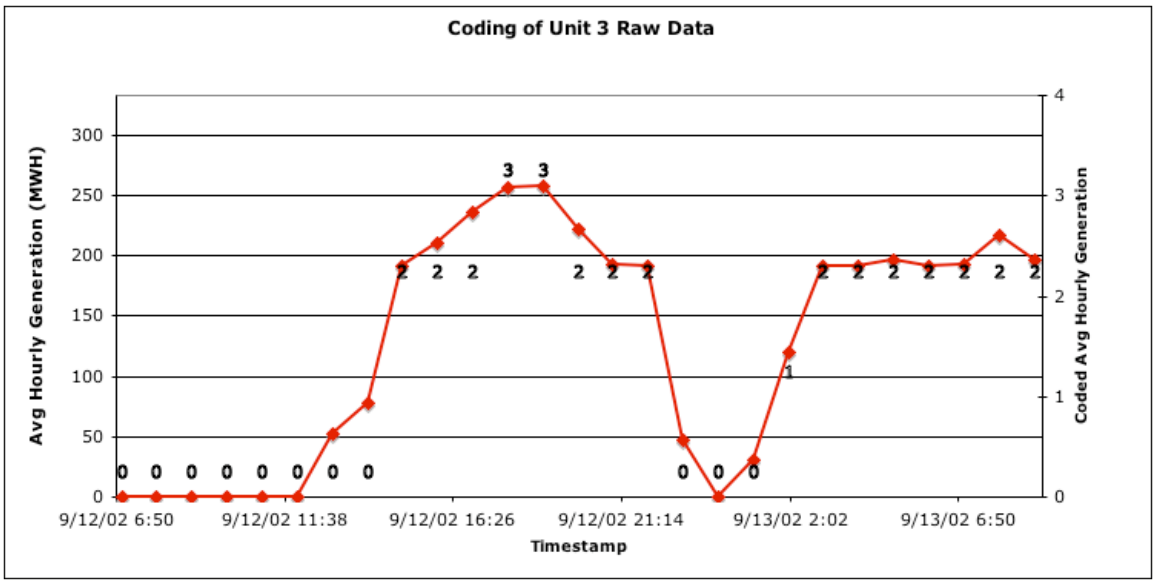

Figure 6.3 – Unit 3 Coding

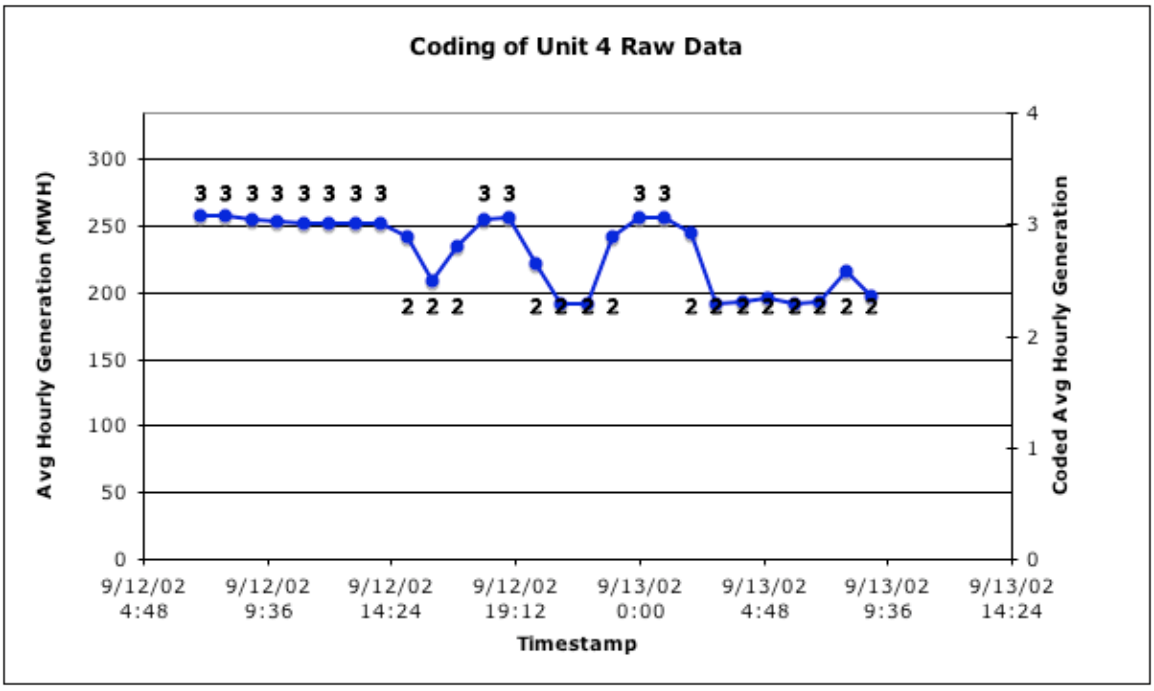

Figure 6.4 – Unit 4 Coding

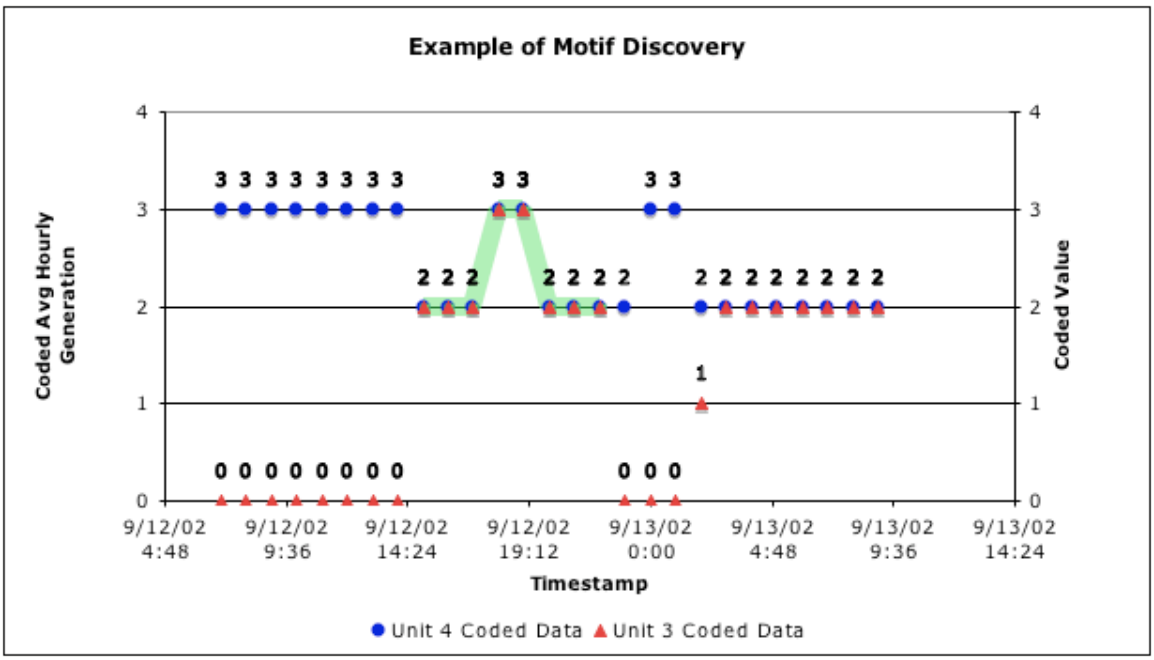

Figure 6.5 –Motif Discovery

Table 6.1 – Coding of Raw Data

	unit 4		unit 3	
	raw value	coded value	raw value	coded value
9/12/02 7:00	258	3	0	0
9/12/02 8:00	257	3	0	0
9/12/02 9:00	254	3	0	0
9/12/02 10:00	253	3	0	0
9/12/02 11:00	252	3	0	0
9/12/02 12:00	252	3	0	0
9/12/02 13:00	252	3	52	0
9/12/02 14:00	252	3	78	0
9/12/02 15:00	241	2	191	2
9/12/02 16:00	209	2	210	2

9/12/02 17:00	235	2	236	2
9/12/02 18:00	255	3	256	3
9/12/02 19:00	256	3	258	3
9/12/02 20:00	222	2	222	2
9/12/02 21:00	191	2	193	2
9/12/02 22:00	191	2	191	2
9/12/02 23:00	242	2	47	0
9/13/02 0:00	256	3	0	0
9/13/02 1:00	256	3	31	0
9/13/02 2:00	244	2	120	1
9/13/02 3:00	191	2	191	2
9/13/02 4:00	192	2	192	2
9/13/02 5:00	196	2	196	2
9/13/02 6:00	191	2	192	2
9/13/02 7:00	193	2	193	2
9/13/02 8:00	216	2	217	2
9/13/02 9:00	197	2	197	2

Rule 1:

If the current sequence is {2,2,2,3,3,2,2,2} and the generating is unit 4,

Then the sequence will be present at generating unit 3 with a confidence of 1.0.

Rule 2:

If the current sequence is {2,2,2,3,3,2,2,2} and the generating is unit 3

Then the sequence will be present at generating unit 4 with a confidence of 0.6.

Note: There are actually two matching sequences in the dataset reviewed, but the variance in the second dataset is zero. A change in sequence values, resulting in a variance greater than zero is desired.

EXPERIMENTAL RESULTS

Discussion of Data Analyzed

As mentioned throughout the document, the data that had been analyzed is power generation data. The data are only generation data from pipeline-natural-gas units. An analysis was performed over both plants and units. Since the data are sampled on a unit-level, each plant is constructed by summing the hourly values for the units within the plant. This data are very sporadic and at time are missing. If the data are missing it is deemed a "don't care" and flagged with a negative one. The Cross Correlation analysis, Principal Component analysis, and Clustering analysis were performed over raw values. The K-Motif discovery was performed over coded data. USD was the coding process, which is a uniform normalization. Each unit or plant's maximum and minimum generation value, non-negative, are used in calculating the bin width and the correct code for each timestamped generation value.

Introduction to Results

The simple description of the analysis is pattern discovery in a highly dimensional data set. Specifically, the idea was to create a global model to reflect power generation by natural gas plants. This global model is intended to include the minimal number of distinct number of characteristics needed to represent power generation from natural gas fired generating units on the power grid. This model is exceptionally useful, but difficult to estimate. The first step to creating the global model is to identify the groups that contain the distinct characteristics. A simple approach to this grouping, or clustering, is to

do a simple overview of the data and assess the possibility of a global model. This research is based on two data sets. The first data set is an average hourly generation data set on a unit-level basis, meaning there exists a generation value for every unit (present at the time) for every timestamp during the dates provided. The second data set is a plant based set, which was computed by summing the output of the generating units within the plant. To grasp the size of this problem one must see the size of the space from which possible patterns can reside. The size of the space is encompassed by N^P where N is the length of the pattern, and P the alphabet size. This is highly advantageous for pattern discovery due, to a larger space from which patterns may be recognized. For instance, there can be 1024 patterns of length four from the alphabet {-1, 0, 1, 2, 3}. Patterns of all the same symbols are interesting, they however add to the total number of patterns to be examined. The total number of possible patterns is worth restricting due to time constraints of a brute-force pattern-matching algorithm. Which is basically a full search of all patterns possible, based upon the codes within alphabet (and any user-defined restrictions). To calculate the total number of searches performed, one may use $U*\left(C^{L_{\max}} - C^{L_{\min}}\right)$; where U represents the number of units or plants, C the number of possible codes, $L_{\max}$ is the maximum length of the sequence, and $L_{\min}$ is the minimum sequence length. For the years of 2002 through 2003, this search space can be over 1631 units or 724 plants. From the previous example this could result in 452,500 searches for patterns of length four for plants, and 1,019,375 searches for patterns of length four for units. Obviously this seems to be a time consuming process, and to reduce the time spent a few eliminations were made. First, patterns with reoccurring symbols that have a frequency greater than 80% were eliminated. Also patterns containing the symbol '-1'

were also excluded. All of the parameters for exclusion mentioned were a configurable component of the software.

Unit Raw Data

The data are, as said earlier, generation data on a per generator basis. This data can be very sparse and especially so with natural gas fired units. So it is common to see long stretches of a signal of zero magnitude, which indicate that a generator is reporting and not producing electricity. There are learning (2002-2003) and testing (2003-2004) data sets. The learning data set contains values for 281 generating units, or 17.2% of the 1631 total units. Of these 281 units the data reported are sparse. Each unit does not necessarily report for every hour of the year. The percentage reported is displayed in figure 7.1 by percentage and the count, or number of units that report with this percentage.

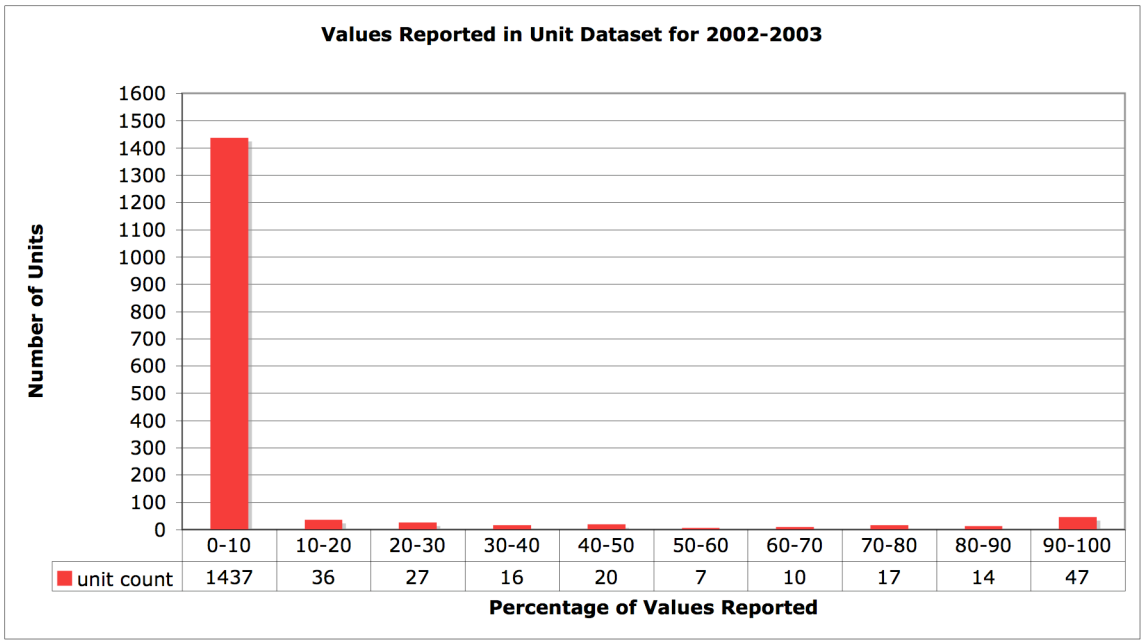

Figure 7.1 – Unit-Level Plot of Learning Data Set Percentages of Values Reported

As is obvious, this data set is very sparse and will present a tough discovery data set. The testing data set contains values for 341 generating units, or 20.8 % of the 1631 total units. The plot, in figure 7.2, is a similar plot for the test data set.

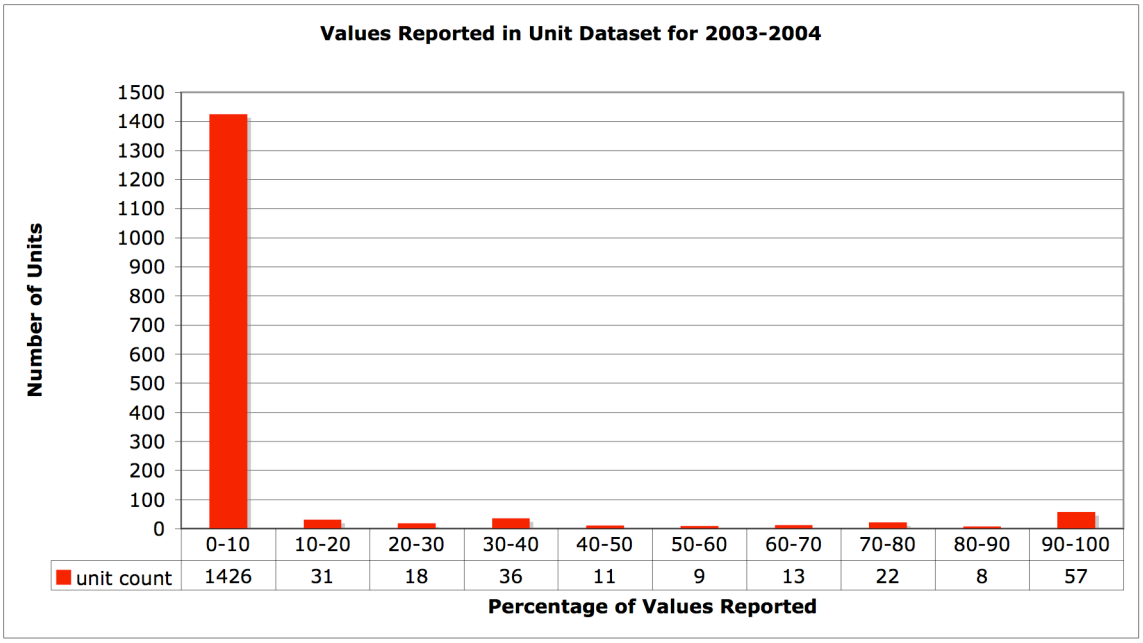

Figure 7.2 – Unit-Level Plot of Testing Data Set Percentages of Values Reported

The author initially thought that the learning data set was a bad data set to start with, but during analysis of the testing data set it was observed that both sets were similar statistically. With this kind of scarcity a Cross Correlation matrix from the entire set will contain many null values, where null values represent missing correlation values. Initially to reduce dimensionality and to aid in cluster recognition, a simple Pearson Cross-Correlation matrix was assembled. Table 7.1 contains the correlations grouped by value (rounded to nearest tenth).

Table 7.1 – Summary of Unit-Level Cross-Correlations

r^2	count
0	2652000
0.01	6362
0.02	150
0.03	8
0.04	8
0.05	2
1	1631

From the raw data, it can be drawn that these statistics from the correlation matrix are feasible. These basic statistics show there exists extreme diversity in the data set where comparing units. With this data one can estimate that finding useful rules may be impossible or at best difficult. These values can be attributed to the market demand. To understand the demand and scarcity of generation values understand the scarcity one must understand that the operation of gas-fired generating units is very unpredictable and is heavily (almost entirely) market controlled.

Principal Component Analysis for Units

It is obvious from the Cross Correlation values above that grouping based on raw value will be challenging. From this point, the researcher attempted to extract useful features from the data set allowing for reduction to aid in an attempt at clustering. Principal Components analysis was performed on the raw data and Figure 7.3 represents the normalized variance of the data set corresponding to each feature, typically called a Screeplot.

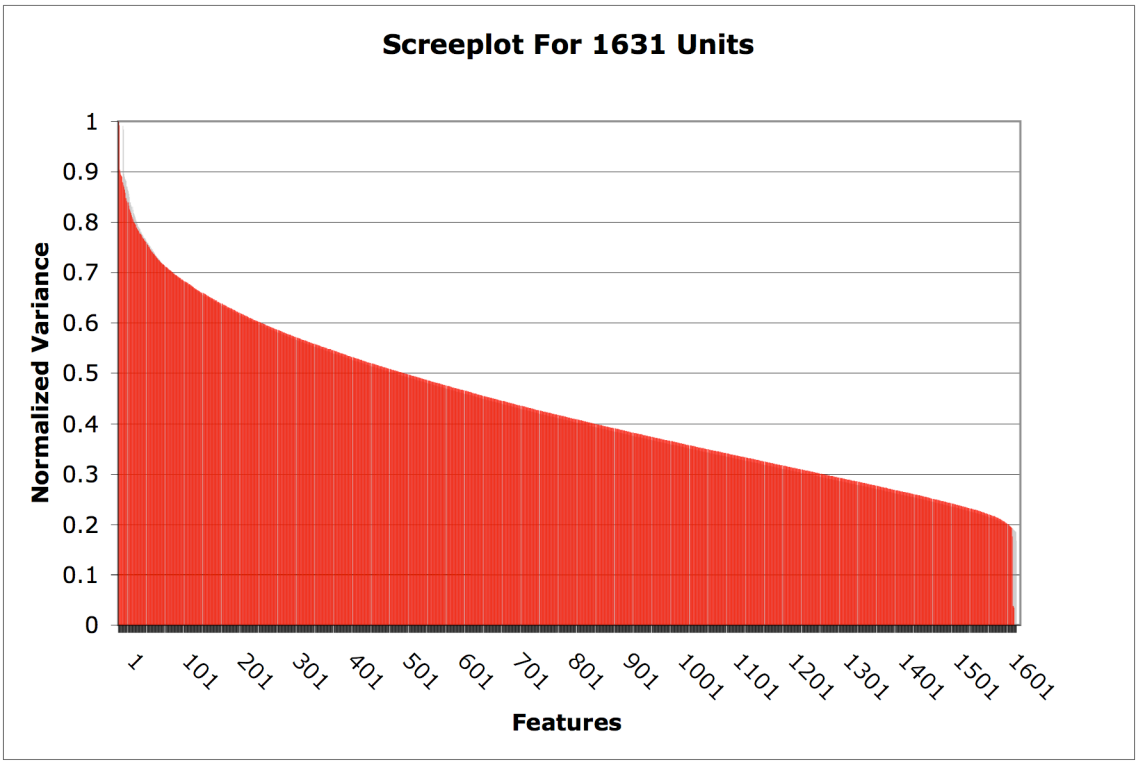

Figure 7.3 – Screeplot for Unit-Level Learning Data Set

At first glance the Screeplot is a bit confusing, and identifying the estimated dimensionality of the data set is best left to Eigenvalue analysis. The Eigenvalues of the result are in fact the variance associated with that dimension, or feature. For this data set a feature represents a unit. To achieve an accurate measure of the maximum number of required dimensions involves identifying the dimensions that encompass 90% of the variance in the data set, according to Kantardzic [12]. For each dimension, a researcher must calculate the percentage of variation represented. For each dimension the percentage variance is calculated as

$$V_p = \frac{V_k}{\sum_{i=0}^{n} V_i} \qquad (7.1)$$

where V_k is the variance associated with cluster k, and the denominator is nothing more than a summation of the variances associated with all features. As noted previously, a typical measure used to identify useful distinct features when analyzing via Principal Component Analysis, is to describe 90% of the variance with the distinct features. To account for 90% of the variance in the learning data set, it would require 1159 units. 1159 Units represents 71% of the total number of units. Obviously this is unfathomable for global modeling, and further leads to a belief that clustering with the learning data set would result in a practically useless model.

K-Means Clustering Analysis for Units

With the PCA results in mind it can be expected that finding the correct number of clusters to operate with would be difficult. To further the clustering concept, the author performed a K-Means clustering analysis with a reduced size from the results of the PCA. Obviously there is no good way to identify features, and as said before a global model with 1159 units is unreasonable. Figure 7.4 displays the within-cluster variation for a clustering analysis using 100 clusters. 100 Clusters was chosen as a random round value that might give an indication of likeness allowing for a large number of clusters. One could reason that allowing more clusters than needed would do nothing more than over-cluster the data. Over-clustering would do nothing more than waste computational time, resulting in many clusters with very low overall errors (wouldn't this be a nice problem to solve!). Since this is not a real time process, wasting computational time can be

dismissed. Low overall cluster error can be easily remedied by reclustering with a low number of clusters. A low overall cluster error for the clusters gives the researcher an optimistic outlook that the clustering analysis is working and can be optimized.

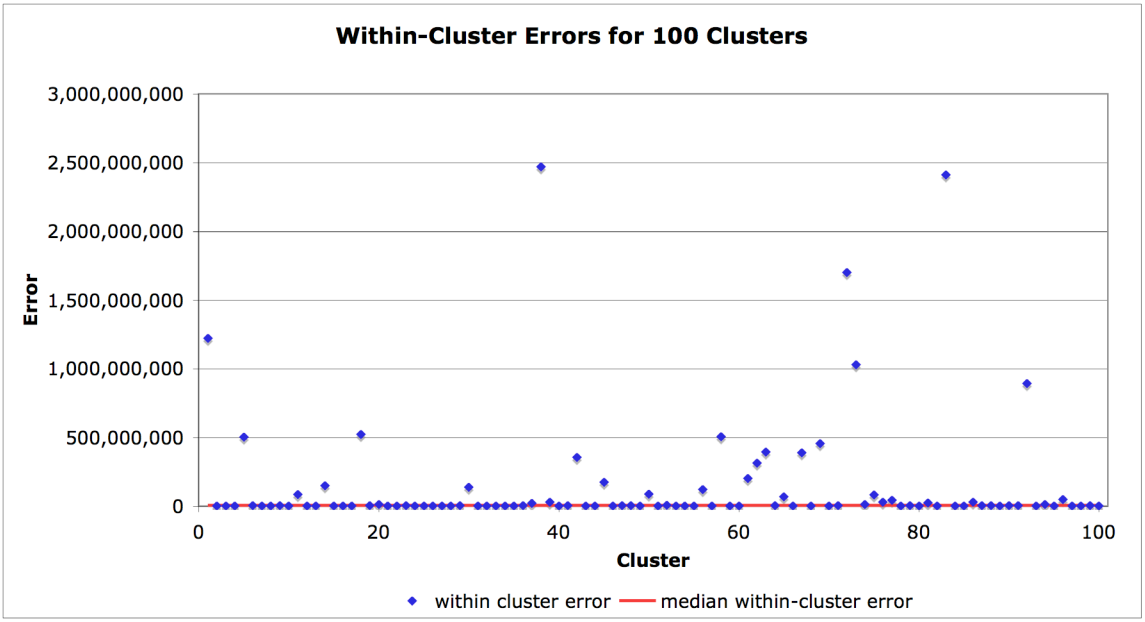

Figure 7.4 – Within-Cluster Error for the Unit-Level Learning Data Set

Table 7.2 contains the basic statistics for the cluster creation.

Table 7.2 – Summary of Within-Cluster Error Statistics for Unit-Level Clustering

Minimum	2,715,492
Maximum	2,471,214,401
Median	4,202,800
Sample Mean	148,377,543
Sample Standard Deviation	422,559,223

Within-cluster variation can be computed via,

$$wcv = \sum_{i=0}^{n-1} d_i^2 \qquad (7.2)$$

Where n represents the number of values in the cluster and d_i represents the distance measure used at the specified index. The distance metric used for the clustering was, typical, Euclidean distance. From the plot above it is easy to see that there are some clusters, of the 100, that have almost no commonness (due to large errors). Furthermore, the mean error of the group is 148,377,543 and the median is 4,202,800. These errors are large when compared to the range of the raw values used to cluster. To conclude, with the results from K-Means clustering analysis along with further confirmation from the principal component analysis, it can be solidified that no reasonable global model is possible from this data set regardless of cluster size or the number of clusters sought.

K-Motif Discovery for Units

The initial focus of this research was to apply pattern recognition techniques and clustering to the problem of energy modeling. From the results of the analysis above, it can be declared that a global model describing the units as a whole would be infeasible. Using the techniques described in the previous section, K-Motifs have been discovered in the energy data. Since the size of the data set would result in a long execution time, due to discovery algorithm choice, a subset of the first 50 units were chosen. The raw data was uniformly encoded prior to the sequence discovery process. The table below is a sample of the full table that is included in the reference that shows the K-Motifs that are

common among units. The table displays the pattern, its support, and the confidence between the two units who share the K-Motif. Large support and confidence values indicate a strong relationship existing between the units. A pattern of large variance is of most importance, but due to operational characteristics of a natural gas-fired generating unit is an unlikely find.

Table 7.3 – Sample of Discovered Sequences Including Support and Confidence Values

unit_1	support_unit_1	unit_2	pattern	conf(unit_2 | unit_1)
1	133	2	2,2,2,2,2,2,3,3	0.8
1	136	2	2,2,2,2,2,3,3	0.79
1	36	2	2,2,2,2,2,3,3,2	0.81
1	99	2	2,2,2,2,2,3,3,3	0.78
1	228	2	2,2,2,2,3	0.82
1	9	2	2,2,2,2,3,2,3	0.67
1	138	2	2,2,2,2,3,3	0.8
1	36	2	2,2,2,2,3,3,2	0.81
1	33	2	2,2,2,2,3,3,2,2	0.76
1	100	2	2,2,2,2,3,3,3	0.78
1	17	2	2,2,2,2,3,3,3,2	0.65
1	83	2	2,2,2,2,3,3,3,3	0.77
1	241	2	2,2,2,3	0.84
1	93	2	2,2,2,3,2	0.76
1	7	2	2,2,2,3,2,2,2,3	1
1	10	2	2,2,2,3,2,3	0.6

1	144	2	2,2,2,3,3	0.81
1	39	2	2,2,2,3,3,2	0.85
1	35	2	2,2,2,3,3,2,2	0.77
1	29	2	2,2,2,3,3,2,2,2	0.79

Once the patterns have been discovered they must be tested on a different set of data. The discovery, or learning, of these patterns was performed on data between 04/01/2002 through 03/31/2003. And the rules were tested over a period between 04/01/2003 and 03/31/2004. The testing results show that K-Motifs exist between many of the tested units. The K-Motifs of interest are those that occur over a long period of time and have some change, and some K-Motifs fitting this requirement have been discovered. If the raw data are analyzed carefully, it is somewhat obvious that interesting K-Motifs will be hard to find in such data. To give an overall description of the success and reliability of the rules the author chose to set a confidence level and require that both learning and testing confidences be greater than or equal to this value. For the unit-level data set, the author required that both confidences be greater than or equal to 0.60, which resulted in 159 rules. Each rule has a confidence value and a support value and high confidences along with high corresponding support values indicate a strong relationship. This relationship and its corresponding rule can be considered the local model for these generating units for this pattern of the K-Motif discovered. The full list of unit-level rules can be found in Appendix 5.

Plant Raw Data

The data for this set is an accumulation of unit hourly data. The accumulation is nothing more than summing all the unit generation values that exist, for a plant and deeming the plant's generation as this summation for the specific timestamp. Unlike the unit data set, the plant data set contains more useful (non-zero) data. 100% of the plants have values, which result in no sparse series. This is due to the fact that for every plant, a unit is operational for every hour during the time sampled. A typical plant series will look little like a unit series, unless there is a single unit in a plant. A plant generation value is all units' generation values summed, and usually has a smaller variance than the units that make up the plant. This will lead to more sequence discoveries (relatively), due to a smoother timeseries, and could allow for more positive and useful results. Table 7.4 lists the cross correlations between plants in a summarized form.

Table 7.4 – Summary of Plant-Level Cross Correlations

Correlation	Count
0	490644
0.1	30670
0.2	1530
0.3	604
0.4	4
1	724

In comparison with the unit-level cross correlation summary, the plant analysis could yield more promise. It is obvious from both Correlation matrices that the data are not

smooth and will only yield results when sequence discovery is performed. Although the plant analysis looks promising by the cross correlation matrix, it does not contain enough correlation clusters to use correlation alone to classify like plants.

Principal Component Analysis for Plants

The same procedure that was used to analyze the unit analysis was also executed for plant data. Unlike the unit-level data set, the plant-level data has a value in both learning and testing data sets for every timestamp expected. The variance was normalized to allow comparison between analyses. As can be seen below the same situation is present in the plant data. There is no definitive “knee”, that is usually present is a Screeplot. The “knee” represents a drastic change in variance covered by features. The “knee” is the point at which the number of significant features, and in this case the number of clusters, would be determined.

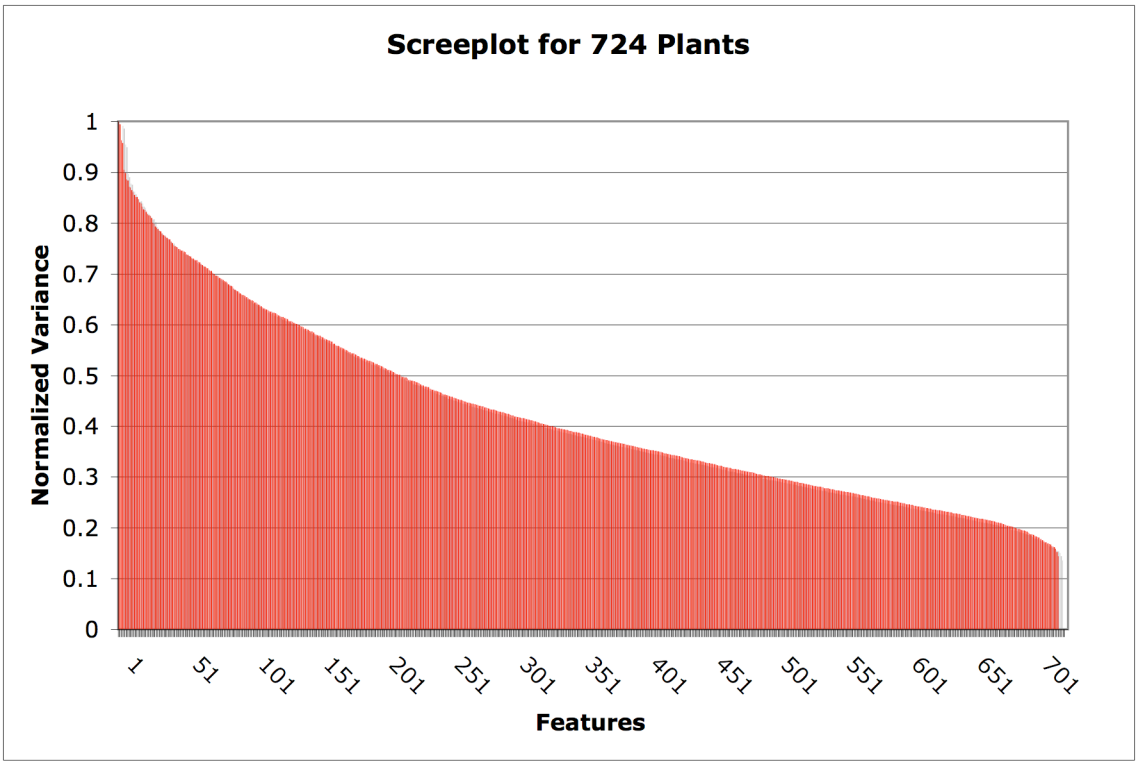

Figure 7.5 – Screeplot for Plant-Level Learning Data Set

As mentioned above the objective of Principal Components analysis is to find the minimum number of features that represent 90% of the variance of the data set. This can be done via the "knee" method or a more precise calculation (formula above). The plant data set requires 580, or 80% of the 724 plants to absorb 90% of the variance. Again a global model with 580 independent variables is cumbersome and useless. The cross dependencies that could arise would hinder the predictive ability of the function if used to form a global model.

K-Means Clustering Analysis for Plants

As in the unit-level clustering analysis, 100 clusters were used due to ambiguity of the Screeplot. From the correlation analysis, one should expect lower within-cluster errors for the plant analysis. Figure 7.6 contains a plot of within-cluster errors.

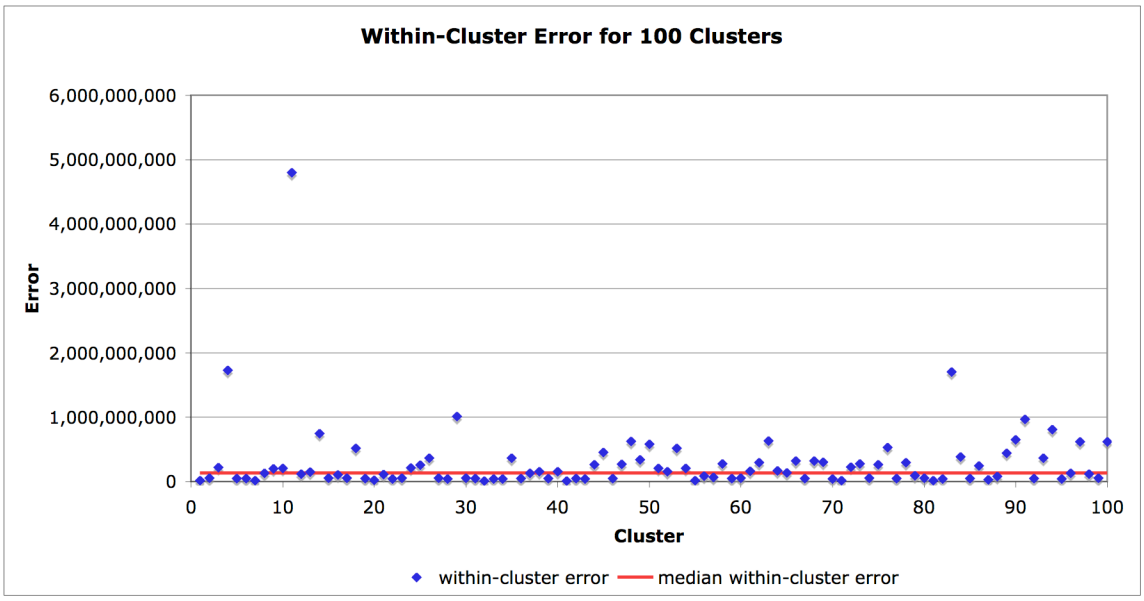

Figure 7.6 – Within-Cluster Error for the Plant-Level Learning Data Set

Table 7.5 lists the basic statistics for within-cluster error for the plant K-Means cluster analysis.

Table 7.5 – Summary of Within-Cluster Error Statistics for Plant-Level Clustering

Minimum	13,490,579
Maximum	4,802,590,406
Median	134,094,691
Sample Mean	288,076,122

Sample Standard Deviation	548,524,774

Although the correlation looked more optimistic the within-cluster error is worse. The median within-cluster error is 33 ½ times larger than the unit median error. All of the basic statistics indicate that the plant analysis is worse for predictive capabilities using K-Means clustering.

K-Motif Discovery for Plants

The K-Motif discovery should allow the researcher to discover any relationships that cannot be observed via means comparison and correlation techniques. The object of diversifying the discovery to include plant level sequences is that with this knowledge a trader can spot plant-level interruptions in power flow and respond using a plant with a matching sequence if appropriate. The plant data set resulted in very few K-Motifs that appeared in both discovery and testing data sets with high confidences. Also the unit analysis standards that were set for the confidence and support values could not be applied to the plant analysis. The support values were extremely low, and resulted in very low confidences in comparison to the unit analysis. The confidence level set for plant-level rules was 0.25, which resulted in 24 valid rules. All association rules for plant-level discoveries can be found in Appendix 6.

Results Discussion

According to the results it can be inferred that unit-level rules provide insight into existing relationships. It is obvious that simple correlation provides no benefit in terms of extracting relationships. With correlations so polarized, values either having very low

correlation or perfect correlation, is shows that little partial similarity exists. The correlation algorithm does not justify matching sequences of large sets where few matches exist. Principal component analysis was proposed as a preliminary to clustering for relationship extraction. The idea was that PCA would amplify the existing relationships and dependencies. Those idealistic relationships weren't obvious as can be seen in the screeplots, which lack the "knee" representation that is common with successful feature extraction analyses; therefore, there are no outstanding features that can be used to create a reasonable number of clusters. From the within-cluster errors displayed in tables 7.5 and 7.2 it can be seen that there exist no useful clusters with the current coding schema. Since neither techniques highlighted features that would be useful in creating a global model to represent power generation, a local model was the next step. The sequence matching performed via K-Motif Brute Force discovery resulted in many relationships that do exist on a local level for a short time. These results can be used to create rules that allow a trader or analyst to predict operation of a unit or plant based on a counterpart of the same type. A full list of the rules associated with plant-level and unit-level analyses reside in Appendices II & III.

CONCLUSION

The problem can be categorized as temporal sequence discovery, meaning discovery of patterns in a timeseries data set. Initially the idea was to propose a global model, one equation defining generation at a specific time for nationwide unit-level generation. This generation was restricted to Pipeline Natural Gas generating units that report to the EPA, which is defined as units that have a capacity greater than 25 MWH. As was discovered a global model was infeasible due to the lack of numerical similarity. This dissimilarity resulted in the creation of a local model consisting of association rules that describe generation based on a subset of the total data. The driver behind the research was to give energy traders the ability to invest in companies with some confidence. The confidence would be based on the high probability that generation patterns will be exhibited by plants or units and are defined by the association rules of the local model. The results of this research are a set of rules that define the local model for Pipeline Natural Gas generating units.

There exist volumes of research dating as far back as the 1970s that are based on the idea of coding and temporal sequence discovery. This allows the researcher to choose techniques that cover a broad range of domains. The domain may however define the proper technique. The group at the University of California at Riverside was inspirational in the research conducted within this Master's thesis. Although the techniques they presented are directed at a specific type of data, they can be adjusted and theories implemented. Eamonn Keogh, at the University of California at Riverside, has introduced

the idea of K-Motifs and “don’t cares”. K-Motifs are motifs that exist in high number that are shared between two sets of timeseries sequences. These sets of timeseries sequences consist of sequences of coded values. The coding of temporal (and non-temporal) sequences has been studied for decades, and are of severe importance when mining sequences. Keogh’s group has contributed to this field, and offer the idea that most timeseries exhibit normality. After being proven false, with relation to this data set, a uniform coding schema has been applied, deemed USD. Once coded, the data was mined for K-Motifs. Once the sequences were discovered their support and confidences were calculated. After having these statistics calculated, the author was able to generate association rules based upon generation patterns. There were no preconceived notions about generation, other than known cycles (in Coal-Fired units); therefore, no patterns were removed from the subset searched other than user-specified flagged patterns. This involved searching the entire space for every possible pattern, which required using a brute-force searching algorithm. Although the software was multi-threaded and optimized for the target language, Java, the software still performed an enormous amount of searches.

There were two simultaneous analyses performed within this Master’s thesis. The first analysis was performed on a unit-level data set where there exists an hourly average generation for every generating unit responsible for reporting to the EPA. The second data set was generated by the author and consists of aggregate values (summations) for all units within a power plant. The original intent of this analysis was to discover unit-level characteristics that would be beneficial, but with further thought it seemed that

traders would possibly be concerned with plant-level characteristics. There were two preliminary steps taken prior to sequence mining, these included Cross Correlation analysis and K-Means clustering Analysis accompanied by Principal Component analysis. The preliminary steps were performed to show that the data was not easily modeled globally. The Cross Correlation analysis was intended to expose the possibility of clustering and to aid in discovery of the number of clusters present in the data set. Both the plant-level and unit-level correlation analyses resulted in very low correlations, which began to confirm that global modeling was infeasible. To aid in choosing the number of clusters to use in K-Means clustering the author performed Principal Components analysis on both data sets. Using a metric provided by Mehmed Kantardzic, which choose the features that represent 90% of the variance, the author was able to gather a minimal feature set from either data set. From the unit-level data, 71% of the total units represented 90% of the variance associated with the unit-level data set. 80% of the plants represented 90% of the variance among the plant-level data set. Both of these values are too large to be used in global modeling, and provide no benefit to clustering analysis. The clustering analysis was performed over 100 features, whether that is plants or units. The idea was to gain insight to see if it was possible to represent the total output using fewer than the entire feature set. From the within-cluster errors above, it can be seen that clustering is fruitless with these data sets. With this information, the author can confirm his previous idea that global modeling is infeasible and inaccurate for this data. To accomplish the local modeling the author performed a K-Motif brute-force discovery over the entire space. To gauge usefulness of the association rules discovered the author set confidence minimums for each rule. For the unit-level data this confidence metric was

set to 0.60, which resulted in 159 rules. The plant-level confidence metric was set to 0.25, which resulted in 24 rules created. All association rules are located in Appendices II and III.

From the previous analysis, it can be drawn that the discovery was a success. There were rules created from discovered K-Motifs that could be used with high confidence. Coding of the data is extremely sensitive to domain and should be studied further. There should be testing performed to find an optimal coding schema for power generation data. A study of this topic would yield fruitful results, which could be applied to coding and other analyses in the power domain. The power domain is extremely interesting as it involves predicting a market that is sensationally dynamic. Smooth timeseries will not be commonly found, and if found of little interest, in power data. For point-wise prediction it seems obvious to apply neural networks to predict this type of highly nonlinear data. The author believes linear techniques along with smoothing and coding would be as effective in prediction of trends, which are of more value to energy traders. With this said, another K-Motif discovery should be performed for each fuel type managed by the EPA. Updates of the previous research should be performed to allow for new rule generation and expired rule removal. All rules generated have a stale factor, meaning all companies change contracts and no rule associated with a company will be permanent. A study of the probability distribution is highly recommended and should be performed prior to any future application of coding schemas. The coding schema should be based on the distribution, as Eamonn Keogh recommended, but as proven the normal assumption cannot be used.

REFERENCES

[1] Hourly Emissions Data. Environmental Protection Agency.

http://www.epa.gov/airmarkets/emissions/raw/

[2] Monitoring Emissions. Environmental Protection Agency.

http://www.epa.gov/airmarkets/monitoring/index.html

[3] Explode Software. Environmental Protection Agency.

http://www.epa.gov/airmarkets/emissions/raw/explode.exe

[4] Electronic Data Reporting Format. Environmental Protection Agency.

http://www.epa.gov/airmarkets/reporting/edr21/index.html

[5] The Plain English Guide to the Clean Air Act. Environmental Protection Agency.

http://www.epa.gov/oar/oaqps/peg_caa/pegcaain.html

[6] EIA Electric Power Forms. Energy Information Administration.

http://www.eia.doe.gov/cneaf/electricity/forms/datamatrix.html

[7] Electric Reliability Council of Texas - ERCOT. North American Electric Reliability Council. http://www.nerc.com/regional/

[8] Christos Stergiou and Dimitrios Siganos . Neural Networks.

http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[9] Distance Section of Cluster Analysis. Statsoft Incorporated.

http://www.statsoft.com/textbook/stcluan.html#d

[10] Oracle SQL* Loader Overview. Oracle Coporation.

http://www.oracle.com/technology/products/database/utilities/htdocs/sql_loader_overview.html

[11] Masters T. Neural, Novel & Hybrid Algorithms for Time Series Prediction. John Wily & Sons. 1995.

[12] Kantardzic M. Data Mining: Concepts, Models, Methods, and Algorithms. Wiley – Institute of Electrical and Electronics Engineers. 2003.

[13] Montgomery D, Runger G. Applied Statistics and Probability for Engineers. John Wiley & Sons. 2002.

[14] Aggarwal C, Han J, Wang J, Yu P. A framework for Projected Clustering of High Dimensional Data Streams. Proceedings of the 30th VLDB Conference. 2004.

[15] Guha S, Mishra N, Motwani R, O'Callaghan L. Clustering Data Streams. IEEE Symposium on Foundations of Computer Science. 2000. Pgs 359-366.

[16] Chiu B, Keogh E, Lonardi S. Probabilistic Discovery of Time Series Motifs. In Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. Washington, D.C. 2003: Pgs 493-498.

[17] Keogh, E, Lonardi S, Chiu B. 2002. Finding Surprising Patterns in a Time Series Database in Linear Time and Space. ACM Special Interest Group on Knowledge Discovery and Data Mining; 2002; Jul 23-26; Edmonton, Alberta (Canada): 11 pgs.

[18] Keogh, Eamonn. A Fast and Robust Method for Pattern Matching in Time Series Databases. 1997.

[19] SCADA. Wikipedia Free Encyclopedia. http://en.wikipedia.org/wiki/SCADA

[20] Böhm C, Kailing K, Kröger P, Zimek A. Computing Clusters of Correlation Connected Objects. Association for Computing Machinery Special Interest Group on Management of Data; 2004; Paris, France: Pgs 455 – 466.

[21] Gupta M, Liu S. Discovery of Conserved Sequence Patterns Using a Stochastic Dictionary Model. American Statistical Association, Journal of the American Statistical Association. 2003;Vol. 98, No. 461, Theory and Methods.

[22] Bellazzi P, Larizza C, De Nicolao G, Riva A, Stefanelli, M. Mining biomedical time series by combining structural analysis and temporal abstractions. Journal of the American Medical Informatics Association; 1998: Pgs 160 – 164.

[23] Yi B, Sidiropoulos N, Johnson T, Jagadish H, Faloutsos C, Biliris A. Online Data Mining for Co-Evolving Sequences. International Conference on Data Engineering. 2000: Pgs 13 – 22.

[24] Van Deursen A, Kuipers T. Identifying Objects using Cluster and Concept Analysis. Association for Computing Machinery, International Conference on Software Engineering. 1999: Pgs 246 – 255.

[25] Torrence C, Compo G. A Practical Guide to Wavelet Analysis. Bulletin of the American Meterological Society. 1998: Vol. 79, No. 1, Pgs 61 – 78.

[26] Wavelet. Wikipedia Free Enclycopedia. http://en.wikipedia.org/wiki/Wavelet.

[27] Lau K, Weng H. Climate Signal Detection Using Wavelet Transform: How to Make a Time Series Sing. Bulletin of the American Meteorological Society. 1995; Vol. 76, No. 12.

[28] Valens, C. 1999. A Really Friendly Guide to Wavelets.

http://perso.orange.fr/polyvalens/clemens/wavelets/wavelets.html

[29] Murguía J, Campos C. Wavelet analysis of chaotic time series. Revista Mexicana De Física. 2005: Vol. 52, No. 2, Pg 155-162.

[30] Hermitian Wavelet. Wikipedia Free Encylcopedia.

http://en.wikipedia.org/wiki/Hermitian_wavelet.

[31] Hermite polynomials. Wikipedia Free Encylcopedia.

http://e.wikipedia.org/wiki/Hermite_polynomial.

[32] Morlet Wavelet. Wikipedia Free Encylcopedia.

http://en.wikipedia.org/wiki/Morlet_wavelet.

[33] Orthogonality. Wikipedia Free Encylcopedia.

http://en.wikipedia.org/wiki/Orthogonal.

[34] Orthogonal Basis. Wolfram Mathworld.

http://mathworld.wolfram.com/OrthogonalBasis.html

[35] Engineering and Statistics Handbook, Spectral Plot. National Institute of Science and Technology (NIST). http://www.itl.nist.gov/div898/handbook/eda/section3/spectrum.htm

[36] Plants – News and Issues – Alabama Power. Alabama Power a Southern Company. http://www.southernco.com/alpower/about/plants.asp?mnuOpco=apc&mnuType=sub&mnuItem=ni

[37] Cotofrei, Paul, & Stoffel, Kilian. Rule Extraction From Time Series Databases Using Classification Trees. Proceedings of International Conference of Applied Informatics. 2002: Pgs 327-332.

[38] Höpper F. Learning Dependencies in Multivariate Time Series. European Conference on Artificial Intelligence. 2002.

[39] Alekseyenko A, Lee C. A stochastic model for creation of genomic sequence blocks.

[40] Lin J, Keogh E, Lonardi S, Patel P. Finding Motifs in Time Series. Proceedings of the Second Workshop on Temporal Data Mining. 2002: Edmonton, Alberta (Canada).

[41] Bussemaker H, Li Hao, Siggia E. Building a dictionary for genomes: Identification of presumptive regulatory sites by statistical analysis. Proceedings of the National Academy of Sciences. 2000: Vol. 97, No. 18.

[42] Mallat S. Theory for Multiresolution Signal Decomposition: The Wavelet Respresentation. Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence. 1989: Vol. 11, No. 7.

APPENDIX I: Source Code

Project: NewKMotif

FileUtilities.java

FileUtilities is a class that exits to load data from a matrix of data into a friendly form usable by other classes. It contains two methods, retrieveOrisplList and retrieveColumnData. The method retrieveOrisplList does nothing more than return a java.util.List object that contains orispls. An orispl is nothing more than a unique identifier, numeric value that represents a distinct plant or unit. The method retriveColumnData fetches data in the form of a java.util.Treemap for a specific orispl from disk. For the non-java audience, a Treemap returns data in order inserted which is crucial when dealing with timestamped data. Although this is inefficient using i/o as a measurement, it performs well when memory is limited and all data cannot be stored in memory (as is typical of large datasets). Both methods are static allowing the ability for easy multithreading and low overhead.

```
package edu.louisville.cecs;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.TreeMap;
import java.util.List;
import java.util.Map;
import org.apache.log4j.Logger;

public class FileUtilities {

  private static final Logger logger = Logger.getLogger( FileUtilities.class );

  public static List retrieveOrisplList( String filename, String inputDelimeter
) {
    /*
     * File format: space,orispl1,orispl2,orispl3,..,orisplN
     * ts1,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
     * ts2,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
     * ts3,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN ..
     * tsn,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
     */
    BufferedReader br = null;
    List orisplList = null;
    try {
      // setup file
```

```
      orisplList = new ArrayList();
      br = new BufferedReader( new FileReader( filename ) );
      // find column with correct orispl
      String [] orisplRow = br.readLine().split( inputDelimeter ); // first row
contains
      // orispls
      for ( int i = 1 ; i < orisplRow.length ; i++ ) {
        Orispl orispl = new Orispl( orisplRow[i] );
        orisplList.add( orispl );
      }
    }
    catch ( FileNotFoundException e ) {
      logger.error( e );
    }
    catch ( IOException e ) {
      logger.error( e );
    }
    finally {
      if ( br != null ) {
        try {
          br.close();
        }
        catch ( IOException e ) {
          logger.error( e );
        }
      }
    }
    return orisplList;
  }

  public static Map retrieveColumnData( String filename, String inputDelimeter,
Orispl orispl ) {
    /*
     * File format: space,orispl1,orispl2,orispl3,..,orisplN
     * ts1,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
     * ts2,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
     * ts3,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN ..
     * tsn,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
     */
    BufferedReader br = null;
    Map dataMap = new TreeMap();
    int timestampColumnNumber = 0;
    try {
      // setup file
      br = new BufferedReader( new FileReader( filename ) );
      // find column with correct orispl
      String [] orisplRow = br.readLine().split( inputDelimeter ); // first row
contains
      // orispls
      int column = 0;
      for ( int i = 0 ; i < orisplRow.length ; i++ ) {
        String field = orisplRow[i];
        if ( field.equals( orispl.toString() ) ) {
```

```
          column = i;
          break;
        }
      }
      // retrieve data for specific orispl
      String line = null;
      SimpleDateFormat sdf = new SimpleDateFormat( "MM/dd/yyyy HH:mm:ss" );
      while ( ( line = br.readLine() ) != null ) {
        String [] row = line.split( inputDelimeter );
        Date timestamp = sdf.parse( row[timestampColumnNumber] );
        Integer value = new Integer( row[column] );
        dataMap.put( timestamp, value );
      }
      logger.info( "RawDataRetriever : orispl " + orispl.toString() + "
contains " + dataMap.values().size()
          + " raw data points." );
    }
    catch ( FileNotFoundException e ) {
      logger.error( e );
    }
    catch ( IOException e ) {
      logger.error( e );
    }
    catch ( ParseException e ) {
      logger.error( e );
    }
    finally {
      if ( br != null ) {
        try {
          br.close();
        }
        catch ( IOException e ) {
          logger.error( e );
        }
      }
    }
    return dataMap;
  }
}
```

Orispl.java

As noted earlier an orispl is nothing more than a unique identifier for plants and units. The orispl class performs basic functions, provided by its methods, which would be expected of an entity representing a unit like storing and accessing raw data, timestamped coded data, and mathes. There are accessors for raw data, and timestamps. Each orispl holds its own raw data and coded raw data, or patterns. The idea of the software is to match coded raw data; therefore, along with basic data it holds matching indices of other orispls (which are denoted by mathcingIndicesMap).

```
package edu.louisville.cecs;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;

import org.apache.log4j.Logger;

public class Orispl {

  private Logger logger = Logger.getLogger( Orispl.class );

  private List patternList = Collections.synchronizedList( new ArrayList() );

  // contains a map of timestamped pattern to an array of matching starting
indices
  private Map matchingIndicesMap = Collections.synchronizedMap( new HashMap( (
int ) ( 0.75 * 3000 ) ) );

  private Map rawDataMap = null; // map contains timestamp, data

  private String orisplId = null;

  private int rawDataSize = 0;

  public Orispl ( String orisplId ) {
    this.orisplId = orisplId;
  }

  public String toString() {
    return orisplId.toString();
  }

  public boolean equals( Object o ) {
    if ( o != null && o instanceof Orispl ) {
      Orispl orispl = ( Orispl ) o;
```

```
      if ( orispl.toString().equals( orisplId ) ) {
        return true;
      }
    }
    return false;
  }

  public int hashCode() {
    // the string will never change
    return orisplId.hashCode();
  }

  /*
   * Raw Data Map Accessors
   */

  public int getRawDataSize() {
    return rawDataSize;
  }

  public Object [] discoverIndices( List patternsList ) {
    Collection c = new ArrayList();
    List sourceList = new ArrayList( rawDataMap.values() );
    int totalLength = sourceList.size();
    int index = -1;
    while ( ( index = Collections.indexOfSubList( sourceList, patternsList ) )
!= -1 ) {
      int value = totalLength - sourceList.size() + index;
      c.add( new Integer( value ) );
      sourceList.subList( 0, index + patternsList.size() ).clear();
    }
    return c.toArray();
  }

  public Integer getRawDataPoint( int idx ) {
    return ( Integer ) rawDataMap.values().toArray()[idx];
  }

  public Map getRawDataSubMap( Date startKey, Date stopKey ) {
    return ( ( TreeMap ) rawDataMap ).subMap( startKey, stopKey ); // must use
sucessor according
    // to the api
  }

  public Date getRawDataKey( int idx ) {
    return ( Date ) rawDataMap.keySet().toArray()[idx];
  }

  public void setRawDataMap( Map dataMap ) {
    rawDataMap = dataMap;
    rawDataSize = rawDataMap.values().size();
  }
```

```
  public void nullifyRawDataMap() {
    rawDataMap = null;
  }

  /*
   * Timestamp Pattern List Accessors
   */

  public Pattern getPatternAt( int index ) {
    return ( Pattern ) patternList.get( index );
  }

  public int indexOfPattern( Pattern p ) {
    return patternList.indexOf( p );
  }

  public void putPattern( Pattern p ) {
    patternList.add( p );
  }

  public int getPatternListSize() {
    return patternList.size();
  }

  /*
   * Pattern Map count accessors
   */

  public void setMatchingIndices( Pattern p, Object [] matchingIndices ) {
    StringBuffer pattern = new StringBuffer();
    for ( int i = 0 ; i < p.getLength() ; i++ ) {
      pattern.append( p.getValue( i ) );
    }
    matchingIndicesMap.put( pattern.toString(), matchingIndices );
    logger.info( "orispl = " + orisplId + " has " + matchingIndices.length + "
instances of {" + p.toString() + "}" );
  }

  public Object [] getMatchingIndices( Pattern p ) {
    StringBuffer pattern = new StringBuffer();
    for ( int i = 0 ; i < p.getLength() ; i++ ) {
      pattern.append( p.getValue( i ) );
    }
    return ( Object [] ) matchingIndicesMap.get( pattern.toString() );
  }
}
```

Pattern.java

A pattern is nothing more than sequence of data. The pattern has no concept of time and is used as a container for a dataset. This class contains accessor methods for length and value(s), while basic toString and equals methods are also implemented. The equals method makes this class unique as it checks for pattern equivalence by checking each value, which is numeric.

```
package edu.louisville.cecs;

import java.util.ArrayList;
import java.util.List;

public class Pattern {

  private List patternList = new ArrayList();

  public Pattern ( Orispl orispl, int startIndex, int finishIndex ) {
    for ( int i = startIndex ; i < finishIndex ; i++ ) {
      patternList.add( orispl.getRawDataPoint( i ) );
    }
  }

  public boolean equals( Object o ) {
    if ( o != null && o instanceof Pattern ) {
      Pattern pattern = ( Pattern ) o;
      // of course the patterns must be the same length
      if ( pattern.getLength() == this.getLength() ) {
        // check values for equivalance
        for ( int i = 0 ; i < this.getLength() ; i++ ) {
          Integer value1 = this.getValue( i );
          Integer value2 = pattern.getValue( i );
          if ( value1.intValue() != value2.intValue() ) {
            return false;
          }
        }
        return true;
      }
    }
    return false;
  }

  public int hashCode() {
    return patternList.hashCode(); // each patternMap should be distinct
  }

  public String toString() {
    StringBuffer sb = new StringBuffer();
    for ( int i = 0 ; i < patternList.size() ; i++ ) {
      String value = ( ( Integer ) patternList.get( i ) ).toString();
      sb.append( value );
      if ( i < patternList.size() - 1 ) {
```

```
        sb.append( "," );
      }
    }
    return sb.toString();
  }

  public int getLength() {
    return patternList.size();
  }

  public Integer getValue( int i ) {
    return ( Integer ) patternList.get( i );
  }
}
```

PatternRunnable.java

As the java folks recognize, this class implements Runnable. This means that this class is the thread. All recognize the run method is the executed block during the thread's life (regardless of condition). The run method locks access to the orisplList preventing deadlock on the list. The isValidPatternLength, isInvalidCodeByPercentage, and containsEnoughMatches are methods that prune matching patterns to criteria set by the user in the property file. They do nothing more than accompany the run method is matching and certification of useful patterns.

```
package edu.louisville.cecs;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;

public class PatternRunnable implements Runnable {

  private Properties properties;

  private List orisplList;

  private List patternList;

  private List threadLockList;

  public PatternRunnable ( Properties properties, List orisplList, List
patternList, List threadLockList ) {
    this.properties = properties;
    this.orisplList = orisplList;
    this.patternList = patternList;
    this.threadLockList = threadLockList;
  }

  public void run() {
    if ( isValidPatternLength() && !isInvalidCodeByPercentage() ) {
      ThreadLock lock = new ThreadLock();
      threadLockList.add( lock );
      try {
        Iterator orisplIterator = orisplList.iterator();
        while ( orisplIterator.hasNext() ) {
          Orispl orispl = ( Orispl ) orisplIterator.next();
          Object [] matchingIndices = orispl.discoverIndices( patternList );
          if ( containsEnoughMatches( matchingIndices.length ) ) {
            //only store one pattern, along with indices to find all instances
of the pattern
            int startIndex = ( ( Integer ) matchingIndices[0] ).intValue();
            int endIndex = startIndex + patternList.size();
            Pattern pattern = new Pattern( orispl, startIndex, endIndex );
            orispl.putPattern( pattern );
            orispl.setMatchingIndices( pattern, matchingIndices );
```

```
          }
        } // end orispl iterator while loop
      } // end try
      finally {
        lock.release();
      }
    } // end if
  }

  private boolean isValidPatternLength() {
    // check pattern length
    int minimumPatternLengthValue = Integer.valueOf( ( String )
properties.getProperty( "minimumPatternLength" ).trim() )
        .intValue();
    int maximumPatternLengthValue = Integer.valueOf( ( String )
properties.getProperty( "maximumPatternLength" ).trim() )
        .intValue();
    if ( patternList.size() < minimumPatternLengthValue || patternList.size() >
maximumPatternLengthValue ) {
      return false;
    }
    return true;
  }

  private boolean isInvalidCodeByPercentage() {
    int minimumPatternValue = Integer.valueOf( ( String )
properties.getProperty( "minimumPattern" ).trim() ).intValue();
    int maximumPatternValue = Integer.valueOf( ( String )
properties.getProperty( "maximumPattern" ).trim() ).intValue();
    for ( int j = minimumPatternValue ; j <= maximumPatternValue ; j++ ) {
      List tempList = new ArrayList( this.patternList );
      int cnt = 0;
      int idx = -1;
      while ( ( idx = tempList.indexOf( new Integer( j ) ) ) != -1 ) {
        cnt++;
        tempList.subList( 0, idx + 1 ).clear(); // trim list
      }
      double saturationPercentage = Double.valueOf(
          ( String ) properties.getProperty( "sequenceSaturationPercentage"
).trim() ).doubleValue();
      if ( ( cnt / ( double ) patternList.size() ) > saturationPercentage ) {
        return true;
      }
    }
    return false;
  }

  private boolean containsEnoughMatches( int size ) {
    int minimumLimit = Integer.valueOf( properties.getProperty(
"minimumNumberOfMatches" ).trim() ).intValue();
    if ( size >= minimumLimit ) {
      return true;
    }
```

```
    return false;
  }
}
```

ThreadLock.java

ThreadLock is nothing more than a lock used during access of the orisplList. This class is typical of many locking mechanisms.

```
package edu.louisville.cecs;

public class ThreadLock {

  private boolean alive;

  public ThreadLock () {
    this.alive = true; // construct and alive
  }

  public void release() {
    this.alive = false;
  }

  public boolean isAlive() {
    return alive;
  }
}
```

TimestampedPattern.java

TimestampedPattern is a class that holds a timestamped pattern. The equals method provides custom equality checking. The isEquivalentPattern is similar to the equals method, the difference is that the logic is not absolute. Not absolute refers to a check for percentage match and time shift. Neither of the options are present in the equals method as it is absolute. Basic pattern details are provided by accessor methods.

```
package edu.louisville.cecs;

import java.util.Date;
import java.util.Map;
import java.util.TreeMap;

import org.apache.log4j.Logger;

public class TimestampedPattern {

  private Map patternMap = new TreeMap();

  private long MILLIS_IN_SECOND = 1000;

  private static final String leadOrLagCount = "lead or lag count";

  private static final String percentageMatch = "percenatage match";

  private int leadOrLagCountValue = 0;

  private double percentageMatchValue = 0.0;

  private Logger logger = Logger.getLogger( TimestampedPattern.class );

  public TimestampedPattern ( Orispl orispl, int startIndex, int endIndex, Map
configMap, String descOfPattern ) {
    try {
      for ( int i = startIndex ; i < endIndex ; i++ ) {
        patternMap.put( orispl.getRawDataKey( i ), orispl.getRawDataPoint( i )
);
      }
    }
    catch ( IndexOutOfBoundsException e ) {
      System.out.println( descOfPattern + " startIndex = " + startIndex + ",
endIndex = " + endIndex
          + " rawdata size = " + orispl.getRawDataSize() );
      logger.error( e );
      System.exit( 0 );
    }
    leadOrLagCountValue = Integer.valueOf( ( String ) configMap.get(
leadOrLagCount ) ).intValue();
    percentageMatchValue = Double.valueOf( ( String ) configMap.get(
percentageMatch ) ).doubleValue();
  }
```

```
  public boolean equals( Object o ) {
    if ( o != null && o instanceof TimestampedPattern ) {
      TimestampedPattern pattern = ( TimestampedPattern ) o;
      if ( leadOrLagCountValue == 0 ) {
        // of course the patterns must be the same length
        if ( pattern.getLength() == this.getLength() ) {
          // check VALUES for equivalance
          for ( int i = 0 ; i < this.getLength() ; i++ ) {
            Integer value1 = this.getValue( i );
            Integer value2 = pattern.getValue( i );
            if ( value1.intValue() != value2.intValue() ) {
              return false;
            }
          }
          // now check KEYS for zero time shift
          for ( int i = 0 ; i < this.getLength() ; i++ ) {
            long ts = ( ( Date ) pattern.getKey( i ) ).getTime();
            long ts2 = ( ( Date ) getKey( i ) ).getTime();
            if ( ( ts - ts2 ) != 0 ) {
              return false;
            }
          }
          return true;
        }
      }
      else {
        return isEquivalentPattern( pattern );
      }
    }
    return false;
  }

  public int hashCode() {
    return patternMap.hashCode(); // each patternMap should be distinct
  }

  public String toString() {
    StringBuffer sb = new StringBuffer();
    for ( int i = 0 ; i < patternMap.values().size() ; i++ ) {
      String value = ( ( Integer ) patternMap.values().toArray()[i]
).toString();
      sb.append( value );
      if ( i < patternMap.values().size() - 1 ) {
        sb.append( "," );
      }
    }
    return sb.toString();
  }

  private boolean isEquivalentPattern( TimestampedPattern pattern ) {
    long timeShift = leadOrLagCountValue * MILLIS_IN_SECOND;
    // of course the patterns must be the same length
```

```
    if ( pattern.getLength() == this.getLength() ) {
      // check VALUES for equivalance
      int matchCount = 0;
      for ( int i = 0 ; i < this.getLength() ; i++ ) {
        Integer s1 = this.getValue( i );
        Integer s2 = pattern.getValue( i );
        if ( s1.equals( s2 ) ) {
          matchCount++;
        }
      }
      double localPercentageMatch = matchCount / this.getLength();
      if ( localPercentageMatch < percentageMatchValue ) {
        return false;
      }
      // now check KEYS for the proper time-stamp shift
      for ( int i = 0 ; i < this.getLength() ; i++ ) {
        long ts = ( ( Date ) pattern.getKey( i ) ).getTime();
        long ts2 = ( ( Date ) getKey( i ) ).getTime();
        if ( ( ts - ts2 ) != timeShift ) {
          return false;
        }
      }
      /*
       * passed all conditions: localPercentageMatch >= globalPercentageMatch ,
of same length
       * (obviously) , and correct time shifting
       */
      return true;
    }
    return false;
  }

  public int getLength() {
    return patternMap.values().size();
  }

  public Date getKey( int idx ) {
    return ( Date ) patternMap.keySet().toArray()[idx];
  }

  public Integer getValue( int idx ) {
    return ( Integer ) patternMap.values().toArray()[idx];
  }
}
```

BruteForceDiscovery.java

This class is the main class that performs all of the coordination among the other classes. This application is a highly-optimized multithreading application. During development there was a high emphasis on developing thread safe code, hopefully that has been achieved! The method kickOffDiscovery performs all thread creation and instantiation and includes call to cross series discovery. The isStillAlive method checks each thread in the pool and removes it if dead (finished) so that kickOffDiscovery can move to the next thread and start another. The method buildPatterns is in charge of building patterns included in the search space (excluding banned patterns). The method isValidPattern is responsible for checking the pattern created to see if it is one of the banned patterns. DiscoverCrossSeriesPatterns method performs a sequence match for every orispl against every other orispl. ClearRawData dumps the raw data for each orispl that has been checked, this method is only used at the end of checking and doesn't add much. GetIntersectionCount returns the intersection count of matching sequences between orispls.

```
package edu.louisville.cecs;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;

import org.apache.log4j.Logger;

import EDU.oswego.cs.dl.util.concurrent.PooledExecutor;

public class BruteForceDiscovery {

  private Properties properties;

  private List orisplList = new ArrayList();

  private List threadLockList = Collections.synchronizedList( new ArrayList()
);

  private static final char outputDelimeter = ';';

  private static final Logger logger = Logger.getLogger(
BruteForceDiscovery.class );
```

```
  private List bannedPatternsList = new ArrayList();

  private int maxNumberOfThreads = 10;

  private PooledExecutor patternDiscoveryThreadPool;

  public static void main( String [] args ) {
    System.out.println( "START: " + new java.util.Date() );
    if ( args.length == 1 ) {
      File configFile = new File( args[0] );
      Properties prop = new Properties();
      try {
        prop.load( new FileInputStream( configFile ) );
        BruteForceDiscovery bfd = new BruteForceDiscovery( prop );
        bfd.kickOffDiscovery();
      }
      catch ( FileNotFoundException e ) {
        e.printStackTrace();
      }
      catch ( IOException e ) {
        e.printStackTrace();
      }
    }
    else {
      System.err.println( "Configuration filename is a required input." );
    }
    System.out.println( "END: " + new java.util.Date() );
  }

  public BruteForceDiscovery ( Properties properties ) {
    // set local properties object
    this.properties = properties;

    // fill the bannedPatternsList
    String [] patterns = properties.getProperty( "bannedPatternList"
).trim().split( "," );
    for ( int i = 0 ; i < patterns.length ; i++ ) {
      String pattern = patterns[i];
      List bannedPattern = new ArrayList();
      for ( int j = 0 ; j < pattern.length() ; j++ ) {
        char c = pattern.charAt( j );
        if ( c == '-' ) {
          String p = Character.toString( c );
          j++;
          p += pattern.charAt( j );
          bannedPattern.add( Integer.valueOf( p ) );
        }
        else {
          bannedPattern.add( Integer.valueOf( Character.toString( c ) ) );
        }
      }
      bannedPatternsList.add( bannedPattern );
```

```
    }

    // fill thread count
    maxNumberOfThreads = Integer.valueOf( properties.getProperty( "numThreads"
).trim() ).intValue();
  }

  public void kickOffDiscovery() {
    Date startDate = new Date( System.currentTimeMillis() );
    patternDiscoveryThreadPool = new PooledExecutor( maxNumberOfThreads );
    patternDiscoveryThreadPool.setMinimumPoolSize( 0 );
    patternDiscoveryThreadPool.waitWhenBlocked();
    int minimumPatternValue = Integer.valueOf( ( String )
properties.getProperty( "minimumPattern" ).trim() ).intValue();
    int maximumPatternValue = Integer.valueOf( ( String )
properties.getProperty( "maximumPattern" ).trim() ).intValue();
    // fill the orispls
    Iterator orisplIterator = FileUtilities.retrieveOrisplList(
        ( String ) properties.getProperty( "inputFilename" ).trim(),
        ( String ) properties.getProperty( "inputDelimeter" ).trim()
).iterator();
    while ( orisplIterator.hasNext() ) {
      Orispl orispl = ( Orispl ) orisplIterator.next();
      orispl.setRawDataMap( FileUtilities.retrieveColumnData( ( String )
properties.getProperty( "inputFilename" )
          .trim(), ( String ) properties.getProperty( "inputDelimeter"
).trim(), orispl ) );
      orisplList.add( orispl );
    }
    for ( int j = minimumPatternValue ; j <= maximumPatternValue ; j++ ) {
      List patternList = new ArrayList();
      patternList.add( new Integer( j ) ); // length one
      if ( isValidPattern( patternList ) ) {
        buildPatterns( patternList );
        PatternRunnable patternRunner = new PatternRunnable( properties,
orisplList, patternList, threadLockList );
        try {
          patternDiscoveryThreadPool.execute( patternRunner );
        }
        catch ( InterruptedException e ) {
          logger.error( e );
        }
      }
    }
    try {
      // wait until all other threads are done
      boolean stillWorking = isStillAlive();
      while ( stillWorking ) {
        logger.info( "sleeping 5 seconds" );
        Thread.sleep( 5000 ); // sleep five seconds
        stillWorking = isStillAlive();
      }
      // free memory
```

```
      clearRawData();
      Date endDate = new Date( System.currentTimeMillis() );
      System.out.println( "Pattern Discovery Start Time = " +
startDate.toString() );
      System.out.println( "Pattern Discovery End Time = " + endDate.toString()
);
      discoverCrossSeriesPatterns();
    }
    catch ( InterruptedException e ) {
      logger.error( e );
    }
  }

  private boolean isStillAlive() {
    for ( int i = 0 ; i < threadLockList.size() ; i++ ) {
      ThreadLock lock = ( ThreadLock ) threadLockList.get( i );
      if ( lock.isAlive() ) {
        logger.info( "thread " + i + " is still alive" );
        return true;
      }
    }
    return false;
  }

  private void buildPatterns( List inPatternList ) {
    int maximumPatternLengthValue = Integer.valueOf( ( String )
properties.getProperty( "maximumPatternLength" ).trim() )
        .intValue();
    if ( inPatternList.size() <= maximumPatternLengthValue ) {
      int minimumPatternValue = Integer.valueOf( ( String )
properties.getProperty( "minimumPattern" ).trim() )
          .intValue();
      int maximumPatternValue = Integer.valueOf( ( String )
properties.getProperty( "maximumPattern" ).trim() )
          .intValue();
      for ( int j = minimumPatternValue ; j <= maximumPatternValue ; j++ ) {
        List patternList = new ArrayList( inPatternList );
        patternList.add( new Integer( j ) );
        // handle pattern that has been passed in
        if ( isValidPattern( patternList ) ) {
          buildPatterns( patternList );
          PatternRunnable patternRunner = new PatternRunnable( properties,
orisplList, patternList, threadLockList );
          try {
            patternDiscoveryThreadPool.execute( patternRunner );
          }
          catch ( InterruptedException e ) {
            logger.error( e );
          }
        }
      }
    }
  }
```

```
  private boolean isValidPattern( List patternList ) {
    Iterator i = bannedPatternsList.iterator();
    while ( i.hasNext() ) {
      if ( Collections.indexOfSubList( patternList, ( List ) i.next() ) != -1 )
{
        return false;
      }
    }
    return true;
  }

  private void discoverCrossSeriesPatterns() {
    Date runDate = new Date( System.currentTimeMillis() );
    SimpleDateFormat sdf = new SimpleDateFormat( "MM/dd/yyy HH:mm:ss" );
    BufferedWriter bw = null;
    int leadOrLag = Integer.valueOf( ( String ) properties.getProperty(
"leadOrLagCount" ).trim() ).intValue();
    int minimumNumberOfMatches = Integer.valueOf( properties.getProperty(
"minimumNumberOfMatches" ) ).intValue();
    System.out.println( "Starting to cross series search" );
    try {
      bw = new BufferedWriter( new FileWriter( "results.scsv" ) );
      bw.write( "Rundate " + outputDelimeter + "Pattern " + outputDelimeter +
"Pattern Length " + outputDelimeter
          + "Orispl1 Id " + outputDelimeter + "Orispl1 Count of Pattern " +
outputDelimeter + "Orispl2 Id "
          + outputDelimeter + "Orispl2 Count of Pattern" + outputDelimeter +
"Intersection Count"
          + System.getProperty( "line.separator" ) );
      for ( int i = 0 ; i < orisplList.size() ; i++ ) {
        Orispl orispl1 = ( Orispl ) orisplList.get( i );
        int start = 0;
        if ( leadOrLag == 0 ) {
          start = i + 1;
        }
        for ( int j = start ; j < orisplList.size() ; j++ ) {
          Orispl orispl2 = ( Orispl ) orisplList.get( j );
          if ( !orispl1.equals( orispl2 ) ) {
            logger.info( "Working on matching orispl = " + orispl1.toString() +
"'s patterns with orispl = "
                + orispl2.toString() + "'s patterns." );
            for ( int k = 0 ; k < orispl1.getPatternListSize() ; k++ ) {
              Pattern pattern1 = orispl1.getPatternAt( k );
              int index = -1;
              int intersectionCount = getIntersectionCount( pattern1, orispl1,
orispl2 );
              if ( ( index = orispl2.indexOfPattern( pattern1 ) ) != -1
                  && intersectionCount > minimumNumberOfMatches ) {
                Pattern pattern2 = orispl2.getPatternAt( index );
                StringBuffer outputBuffer = new StringBuffer();
                // print rundate
                outputBuffer.append( sdf.format( runDate ) );
```

```
                // print pattern
                outputBuffer.append( outputDelimeter );
                outputBuffer.append( "{" + pattern1.toString() + "}" );
                // print length of pattern
                outputBuffer.append( outputDelimeter );
                outputBuffer.append( pattern1.getLength() );
                // print orispl1's id
                outputBuffer.append( outputDelimeter );
                outputBuffer.append( orispl1.toString() );
                // print orispl1's count of pattern
                outputBuffer.append( outputDelimeter );
                outputBuffer.append( orispl1.getMatchingIndices( pattern1
).length );
                // print orispl2's id
                outputBuffer.append( outputDelimeter );
                outputBuffer.append( orispl2.toString() );
                // print orispl2's count of pattern
                outputBuffer.append( outputDelimeter );
                outputBuffer.append( orispl2.getMatchingIndices( pattern2
).length );
                // print intersection count
                outputBuffer.append( outputDelimeter );
                outputBuffer.append( intersectionCount );
                bw.write( outputBuffer.toString() + System.getProperty(
"line.separator" ) );
              } // end good patttern if
            } // end pattern loop
          } // end good orispl if
        } // end list of orispls
      }
    }
    catch ( IOException e ) {
      logger.error( e );
    }
    finally {
      if ( bw != null ) {
        try {
          bw.close();
        }
        catch ( IOException e ) {
          logger.error( e );
        }
      }
    }
  }

  private void clearRawData() {
    Iterator i = orisplList.iterator();
    while ( i.hasNext() ) {
      ( ( Orispl ) i.next() ).nullifyRawDataMap();
    }
  }
```

```
  private int getIntersectionCount( Pattern pattern, Orispl orispl1, Orispl
orispl2 ) {
    Object [] orispl1MatchingIndices = orispl1.getMatchingIndices( pattern );
    Object [] orispl2MatchingIndices = orispl2.getMatchingIndices( pattern );
    int intersectionCount = 0;

    if ( orispl1MatchingIndices != null && orispl2MatchingIndices != null ) {
      if ( orispl1MatchingIndices.length < orispl2MatchingIndices.length ) {
        List orispl2MatchingIndicesList = Arrays.asList( orispl2MatchingIndices
);
        for ( int i = 0 ; i < orispl1MatchingIndices.length ; i++ ) {
          Integer orispl1Index = ( Integer ) orispl1MatchingIndices[i];
          if ( orispl2MatchingIndicesList.contains( orispl1Index ) ) {
            intersectionCount++;
          }
        }
      }
      else {
        List orispl1MatchingIndicesList = Arrays.asList( orispl1MatchingIndices
);
        for ( int i = 0 ; i < orispl2MatchingIndices.length ; i++ ) {
          Integer orispl2Index = ( Integer ) orispl2MatchingIndices[i];
          if ( orispl1MatchingIndicesList.contains( orispl2Index ) ) {
            intersectionCount++;
          }
        }
      }
    }
    return intersectionCount;
  }
}
```

Project: Pattern-Tester

FileUtilities.java

FileUtilities is a class that exits to load data from a matrix of data into a friendly form usable by other classes. It contains two methods, retrieveOrisplList and retrieveColumnData. The method retrieveOrisplList does nothing more than return a java.util.List object that contains orispls. An orispl is nothing more than a unique identifier, numeric value that represents a distinct plant or unit. The method retriveColumnData fetches data in the form of a java.util.Treemap for a specific orispl from disk. For the non-java audience, a Treemap returns data in order inserted which is crucial when dealing with timestamped data. Although this is inefficient using i/o as a measurement, it performs well when memory is limited and all data cannot be stored in memory (as is typical of large datasets). Both methods are static allowing the ability for easy multithreading and low overhead.

```
package edu.louisville.cecs;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.TreeMap;
import java.util.List;
import java.util.Map;
import org.apache.log4j.Logger;

public class FileUtilities {

    private static final Logger logger = Logger.getLogger( FileUtilities.class
);

    public static List retrieveOrisplList( String filename, String
inputDelimeter ) {
        /*
         * File format: space,orispl1,orispl2,orispl3,..,orisplN
         * ts1,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
         * ts2,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
         * ts3,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN ..
         * tsn,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
         */
        BufferedReader br = null;
        List orisplList = null;
        try {
            // setup file
            orisplList = new ArrayList();
            br = new BufferedReader( new FileReader( filename ) );
```

```
            // find column with correct orispl
            String [] orisplRow = br.readLine().split( inputDelimeter ); //
first row contains
            // orispls
            for ( int i = 1 ; i < orisplRow.length ; i++ ) {
                Orispl orispl = new Orispl( orisplRow[i] );
                orisplList.add( orispl );
            }
        }
        catch ( FileNotFoundException e ) {
            logger.error( e );
        }
        catch ( IOException e ) {
            logger.error( e );
        }
        finally {
            if ( br != null ) {
                try {
                    br.close();
                }
                catch ( IOException e ) {
                    logger.error( e );
                }
            }
        }
        return orisplList;
    }

    public static Map retrieveColumnData( String filename, String
inputDelimeter, Orispl orispl ) {
        /*
         * File format: space,orispl1,orispl2,orispl3,..,orisplN
         * ts1,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
         * ts2,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
         * ts3,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN ..
         * tsn,valueOrispl1,valueOrispl2,valueOrispl3..,valueOrisplN
         */
        BufferedReader br = null;
        Map dataMap = new TreeMap();
        int timestampColumnNumber = 0;
        try {
            // setup file
            br = new BufferedReader( new FileReader( filename ) );
            // find column with correct orispl
            String [] orisplRow = br.readLine().split( inputDelimeter ); //
first row contains
            // orispls
            int column = 0;
            for ( int i = 0 ; i < orisplRow.length ; i++ ) {
                String field = orisplRow[i];
                if ( field.equals( orispl.toString() ) ) {
                    column = i;
                    break;
```

```
                }
            }
            // retrieve data for specific orispl
            String line = null;
            SimpleDateFormat sdf = new SimpleDateFormat( "MM/dd/yyyy HH:mm:ss"
);
            while ( ( line = br.readLine() ) != null ) {
                String [] row = line.split( inputDelimeter );
                Date timestamp = sdf.parse( row[timestampColumnNumber] );
                Integer value = new Integer( row[column] );
                dataMap.put( timestamp, value );
            }
            logger.info( "RawDataRetriever : orispl " + orispl.toString() + "
contains " + dataMap.values().size()
                    + " raw data points." );
        }
        catch ( FileNotFoundException e ) {
            logger.error( e );
        }
        catch ( IOException e ) {
            logger.error( e );
        }
        catch ( ParseException e ) {
            logger.error( e );
        }
        finally {
            if ( br != null ) {
                try {
                    br.close();
                }
                catch ( IOException e ) {
                    logger.error( e );
                }
            }
        }
        return dataMap;
    }
}
```

Orispl.java

As noted earlier an orispl is nothing more than a unique identifier for plants and units. The orispl class performs basic functions, provided by its methods, which would be expected of an entity representing a unit like storing and accessing raw data, timestamped coded data, and mathes. There are accessors for raw data, and timestamps. Each orispl holds its own raw data and coded raw data, or patterns. The idea of the software is to match coded raw data; therefore, along with basic data it holds matching indices of other orispls (which are denoted by mathcingIndicesMap).

```
package edu.louisville.cecs;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.Map;

public class Orispl {

    private List patternList = Collections.synchronizedList( new ArrayList() );

    private Map rawDataMap = null; // map contains timestamp, data

    private String orisplId = null;

    private int rawDataSize = 0;

    public int getTimestampedPatternListSize() {
        return patternList.size();
    }

    public void nullifyRawDataMap() {
        rawDataMap = null; // not needed anymore
    }

    public Orispl ( String orisplId ) {
        this.orisplId = orisplId;
    }

    public void putPattern( TimestampedPattern tp ) {
        synchronized ( patternList ) {
            patternList.add( tp );
        }
    }

    public String toString() {
        return orisplId.toString();
    }

    public boolean equals( Object o ) {
        if ( o != null && o instanceof Orispl ) {
```

```
            Orispl orispl = ( Orispl ) o;
            if ( orispl.toString().equals( orisplId ) ) {
                return true;
            }
        }
        return false;
    }

    // implement hashcode
    public int hashCode() {
        // the string will never change
        return orisplId.hashCode();
    }

    public int getRawDataSize() {
        return rawDataSize;
    }

    // only used for list comparisons
    public List getRawDataList() {
        return new ArrayList( rawDataMap.values() );
    }

    // should be used
    public Integer getRawDataPoint( int idx ) {
        return ( Integer ) rawDataMap.values().toArray()[idx];
    }

    public TimestampedPattern getPatternAt( int index ) {
        return ( TimestampedPattern ) patternList.get( index );
    }

    public Date getRawDataKey( int idx ) {
        return ( Date ) rawDataMap.keySet().toArray()[idx];
    }

    public void setRawDataMap( Map dataMap ) {
        rawDataMap = dataMap;
        rawDataSize = rawDataMap.values().size();
    }
}
```

Tester.java

Tester is the main class for this application. It performs the act of testing to see if a pattern exists in the provided dataset. The patterns are read from a file, as is the dataset. It is used to check to see if old sequences exist in a new dataset. The startSearch method performs the work while using buildPatternList to create the pattern list from the file.

```
package edu.louisville.cecs;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.TreeMap;


public class Tester {

    public static void main( String [] args ) {
        if ( args.length == 1 ) {
            Tester tester = new Tester();
            tester.startSearch( args[0] );
        }
        else {
            System.out.println( "need to input configuration filename" );
        }
    }

    public void startSearch( String propertyFilename ) {
        Properties properties = new Properties();
        try {
            properties.load( new FileInputStream( new File( propertyFilename )
) );
            String fileName = properties.getProperty( "inputFilename" );
            String delimeter = properties.getProperty( "inputDelimeter" );
            String orisplId = properties.getProperty( "orisplId" );
            Orispl orispl = new Orispl( orisplId );
            TreeMap rawDataMap = ( TreeMap ) FileUtilities.retrieveColumnData(
fileName, delimeter, orispl );
            Object [] keys = rawDataMap.keySet().toArray();
            List values = new ArrayList( rawDataMap.values() );
            int length = values.size();
            String patternDelimeter = properties.getProperty(
"patternDelimeter" ).replaceAll( " ", "" );
            String pattern = properties.getProperty( "pattern" ).replaceAll( "
", "" );
            List patternList = buildPatternList( patternDelimeter, pattern );
            int index = -1;
```

```
            StringBuffer buffer = new StringBuffer();
            List sourceList = new ArrayList( values );
            while ( ( index = Collections.indexOfSubList( sourceList,
patternList ) ) != -1 ) {
                int diff = length - sourceList.size();
                Object startKey = keys[index + diff];
                Object stopKey = keys[index + patternList.size() + diff];
                Map m = rawDataMap.subMap( startKey, stopKey );
                buffer.append( printMap( m ) );
                sourceList.subList( 0, index + 1 ).clear(); // chop list
            }
            System.out.println( buffer.toString() );
        }
        catch ( FileNotFoundException e ) {
            e.printStackTrace();
        }
        catch ( IOException e ) {
            e.printStackTrace();
        }
    }

    private List buildPatternList( String delimeter, String pattern ) {
        String [] fieldsOfPattern = pattern.split( delimeter );
        List patternList = new ArrayList();
        for ( int i = 0 ; i < fieldsOfPattern.length ; i++ ) {
            patternList.add( Integer.valueOf( fieldsOfPattern[i] ) );
        }
        return patternList;
    }

    private StringBuffer printMap( Map m ) {
        StringBuffer buffer = new StringBuffer();
        Object [] keys = m.keySet().toArray();
        Object [] values = m.values().toArray();
        for ( int i = 0 ; i < keys.length ; i++ ) {
            Object key = keys[i];
            Object value = values[i];
            buffer.append( "[" );
            buffer.append( key );
            buffer.append( "," );
            buffer.append( value );
            buffer.append( "]" );
        }
        return buffer.append( System.getProperty( "line.separator" ) );
    }

}
```

TimestampedPattern.java

TimestampedPattern is a class that holds a timestamped pattern. The equals method provides custom equality checking. The isEquivalentPattern is similar to the equals method, the difference is that the logic is not absolute. Not absolute refers to a check for percentage match and time shift. Neither of the options are present in the equals method as it is absolute. Basic pattern details are provided by accessor methods.

```
package edu.louisville.cecs;

import java.util.Date;
import java.util.Map;
import java.util.TreeMap;

public class TimestampedPattern {

    private Map patternMap = new TreeMap();

    public TimestampedPattern ( Orispl orispl, int start, int patternLength ) {
        for ( int i = start ; i < start + patternLength ; i++ ) {
            Date key = orispl.getRawDataKey( i );
            Integer value = orispl.getRawDataPoint( i );
            patternMap.put( key, value );
        }
    }

    public boolean equals( Object o ) {
        if ( o != null && o instanceof TimestampedPattern ) {
            TimestampedPattern pattern = ( TimestampedPattern ) o;
            // of course the patterns must be the same length
            if ( pattern.getLength() == this.getLength() ) {
                // check VALUES for equivalance
                for ( int i = 0 ; i < this.getLength() ; i++ ) {
                    Integer value1 = this.getValue( i );
                    Integer value2 = pattern.getValue( i );
                    if ( value1.intValue() != value2.intValue() ) {
                        return false;
                    }
                }
                // now check KEYS for zero time shift
                for ( int i = 0 ; i < this.getLength() ; i++ ) {
                    long ts = ( ( Date ) pattern.getKey( i ) ).getTime();
                    long ts2 = ( ( Date ) getKey( i ) ).getTime();
                    if ( ( ts - ts2 ) != 0 ) {
                        return false;
                    }
                }
                return true;
            }
        }
        return false;
    }
```

```
    public int hashCode() {
        return patternMap.hashCode(); // each patternMap should be distinct
    }

    public String toString() {
        StringBuffer sb = new StringBuffer();
        for ( int i = 0 ; i < patternMap.values().size() ; i++ ) {
            if ( i != 0 && ( i != patternMap.values().size() ) ) {
                sb.append( "," );
            }
            String value = ( ( Integer ) patternMap.values().toArray()[i]
).toString();
            sb.append( value );
        }
        return sb.toString();
    }

    public boolean isEquivalentPattern( TimestampedPattern pattern, long
timeShift, double globalPercentageMatch ) {
        // of course the patterns must be the same length
        if ( pattern.getLength() == this.getLength() ) {
            // check VALUES for equivalance
            int matchCount = 0;
            for ( int i = 0 ; i < this.getLength() ; i++ ) {
                Integer s1 = this.getValue( i );
                Integer s2 = pattern.getValue( i );
                if ( s1.equals( s2 ) ) {
                    matchCount++;
                }
            }
            double localPercentageMatch = matchCount / this.getLength();
            if ( localPercentageMatch < globalPercentageMatch ) {
                return false;
            }
            // now check KEYS for the proper time-stamp shift
            for ( int i = 0 ; i < this.getLength() ; i++ ) {
                long ts = ( ( Date ) pattern.getKey( i ) ).getTime();
                long ts2 = ( ( Date ) getKey( i ) ).getTime();
                if ( ( ts - ts2 ) != timeShift ) {
                    return false;
                }
            }
            /*
             * passed all conditions: localPercentageMatch >=
globalPercentageMatch , of same length
             * (obviously) , and correct time shifting
             */
            return true;
        }
        return false;
    }
```

```
    public int getLength() {
        return patternMap.values().size();
    }

    public Date getKey( int idx ) {
        return ( Date ) patternMap.keySet().toArray()[idx];
    }

    public Integer getValue( int idx ) {
        return ( Integer ) patternMap.values().toArray()[idx];
    }

}
```

APPENDIX II: Unit-Level Association Rules

				learning	testing
unit_1	support_unit_1	unit_2	pattern	conf(u1 |u2)	conf(u2 | u1)
1	36	2	2,2,2,2,2,3,3,2	0.81	0.6
1	228	2	2,2,2,2,3	0.82	0.61
1	138	2	2,2,2,2,3,3	0.8	0.62
1	36	2	2,2,2,2,3,3,2	0.81	0.65
1	33	2	2,2,2,2,3,3,2,2	0.76	0.62
1	83	2	2,2,2,2,3,3,3,3	0.77	0.64
1	241	2	2,2,2,3	0.84	0.64
1	7	2	2,2,2,3,2,2,2,3	1	0.67
1	144	2	2,2,2,3,3	0.81	0.65
1	39	2	2,2,2,3,3,2	0.85	0.66
1	35	2	2,2,2,3,3,2,2	0.77	0.62
1	29	2	2,2,2,3,3,2,2,2	0.79	0.64
1	104	2	2,2,2,3,3,3	0.79	0.64
1	87	2	2,2,2,3,3,3,3	0.78	0.69
1	82	2	2,2,2,3,3,3,3,3	0.79	0.68
1	268	2	2,2,3	0.82	0.7
1	104	2	2,2,3,2	0.75	0.61
1	92	2	2,2,3,2,2	0.74	0.6
1	7	2	2,2,3,2,2,2,3	1	1
1	9	2	2,2,3,2,2,3	0.67	0.67
1	161	2	2,2,3,3	0.79	0.7
1	44	2	2,2,3,3,2	0.82	0.65
1	41	2	2,2,3,3,2,2	0.78	0.61
1	36	2	2,2,3,3,2,2,2	0.72	0.63
1	35	2	2,2,3,3,2,2,2,2	0.71	0.65
1	114	2	2,2,3,3,3	0.77	0.73
1	95	2	2,2,3,3,3,3	0.78	0.78
1	89	2	2,2,3,3,3,3,3	0.8	0.76
1	78	2	2,2,3,3,3,3,3,3	0.78	0.85
1	114	2	2,3,2	0.77	0.61
1	7	2	2,3,2,2,2,3	1	1
1	9	2	2,3,2,2,3	0.67	0.64
1	10	2	2,3,2,3	0.6	0.64
1	178	2	2,3,3	0.79	0.74
1	51	2	2,3,3,2	0.78	0.64
1	46	2	2,3,3,2,2	0.78	0.64
1	41	2	2,3,3,2,2,2	0.76	0.66
1	40	2	2,3,3,2,2,2,2	0.73	0.67
1	37	2	2,3,3,2,2,2,2,2	0.73	0.64
1	126	2	2,3,3,3	0.78	0.79
1	16	2	2,3,3,3,2,2,2,2	0.63	0.63
1	106	2	2,3,3,3,3	0.78	0.85
1	16	2	2,3,3,3,3,3,3,2	0.69	0.89

1	268	2	3,2,2	0.84	0.69
1	243	2	3,2,2,2	0.85	0.69
1	230	2	3,2,2,2,2	0.84	0.65
1	12	2	3,2,2,2,3	1	0.83
1	7	2	3,2,2,2,3,2	0.86	0.6
1	25	2	3,2,3	0.68	0.64
1	177	2	3,3,2	0.85	0.77
1	162	2	3,3,2,2	0.83	0.73
1	146	2	3,3,2,2,2	0.86	0.74
1	141	2	3,3,2,2,2,2	0.85	0.7
1	136	2	3,3,2,2,2,2,2	0.85	0.66
1	135	2	3,3,2,2,2,2,2,2	0.84	0.63
1	15	2	3,3,2,3	0.73	0.64
1	125	2	3,3,3,2	0.81	0.77
1	113	2	3,3,3,2,2	0.81	0.73
1	104	2	3,3,3,2,2,2	0.83	0.72
1	100	2	3,3,3,2,2,2,2	0.82	0.65
1	98	2	3,3,3,2,2,2,2,2	0.83	0.63
1	105	2	3,3,3,3,2	0.83	0.77
1	96	2	3,3,3,3,2,2	0.81	0.72
1	88	2	3,3,3,3,2,2,2	0.84	0.69
1	84	2	3,3,3,3,2,2,2,2	0.83	0.61
1	87	2	3,3,3,3,3,2,2	0.8	0.67
1	81	2	3,3,3,3,3,2,2,2	0.83	0.63
1	7	2	3,3,3,3,3,2,3,2	0.86	1
1	76	2	3,3,3,3,3,3,2,2	0.79	0.78
2	145	1	2,2,2,2,2,2,3,3	0.74	0.79
2	147	1	2,2,2,2,2,3,3	0.73	0.79
2	41	1	2,2,2,2,2,3,3,2	0.71	0.72
2	106	1	2,2,2,2,2,3,3,3	0.73	0.83
2	242	1	2,2,2,2,3	0.77	0.8
2	154	1	2,2,2,2,3,3	0.72	0.84
2	42	1	2,2,2,2,3,3,2	0.69	0.8
2	36	1	2,2,2,2,3,3,2,2	0.69	0.81
2	110	1	2,2,2,2,3,3,3	0.71	0.86
2	89	1	2,2,2,2,3,3,3,3	0.72	0.86
2	258	1	2,2,2,3	0.78	0.83
2	92	1	2,2,2,3,2	0.77	0.71
2	9	1	2,2,2,3,2,2,2,3	0.78	0.67
2	160	1	2,2,2,3,3	0.73	0.86
2	46	1	2,2,2,3,3,2	0.72	0.83
2	38	1	2,2,2,3,3,2,2	0.71	0.86
2	34	1	2,2,2,3,3,2,2,2	0.68	0.85
2	114	1	2,2,2,3,3,3	0.72	0.88
2	93	1	2,2,2,3,3,3,3	0.73	0.88
2	86	1	2,2,2,3,3,3,3,3	0.76	0.95
2	280	1	2,2,3	0.79	0.84

2	102	1	2,2,3,2	0.76	0.75
2	87	1	2,2,3,2,2	0.78	0.71
2	9	1	2,2,3,2,2,2,3	0.78	0.67
2	7	1	2,2,3,2,2,3	0.86	0.6
2	174	1	2,2,3,3	0.73	0.84
2	53	1	2,2,3,3,2	0.68	0.74
2	47	1	2,2,3,3,2,2	0.68	0.77
2	41	1	2,2,3,3,2,2,2	0.63	0.76
2	40	1	2,2,3,3,2,2,2,2	0.63	0.76
2	121	1	2,2,3,3,3	0.73	0.89
2	99	1	2,2,3,3,3,3	0.75	0.9
2	92	1	2,2,3,3,3,3,3	0.77	0.89
2	11	1	2,2,3,3,3,3,3,2	0.82	0.67
2	81	1	2,2,3,3,3,3,3,3	0.75	0.88
2	114	1	2,3,2	0.77	0.74
2	103	1	2,3,2,2	0.78	0.71
2	93	1	2,3,2,2,2	0.76	0.69
2	9	1	2,3,2,2,2,3	0.78	0.67
2	8	1	2,3,2,2,3	0.75	0.7
2	198	1	2,3,3	0.71	0.85
2	61	1	2,3,3,2	0.66	0.73
2	55	1	2,3,3,2,2	0.65	0.79
2	50	1	2,3,3,2,2,2	0.62	0.79
2	47	1	2,3,3,2,2,2,2	0.62	0.81
2	43	1	2,3,3,2,2,2,2,2	0.63	0.76
2	136	1	2,3,3,3	0.72	0.88
2	112	1	2,3,3,3,3	0.74	0.9
2	13	1	2,3,3,3,3,3,2	0.77	0.6
2	15	1	2,3,3,3,3,3,3,2	0.73	1
2	284	1	3,2,2	0.79	0.82
2	261	1	3,2,2,2	0.79	0.82
2	244	1	3,2,2,2,2	0.79	0.81
2	16	1	3,2,2,2,3	0.75	0.77
2	9	1	3,2,2,2,3,2	0.67	0.6
2	20	1	3,2,2,3	0.7	0.71
2	199	1	3,3,2	0.75	0.85
2	178	1	3,3,2,2	0.76	0.84
2	165	1	3,3,2,2,2	0.76	0.85
2	157	1	3,3,2,2,2,2	0.76	0.85
2	150	1	3,3,2,2,2,2,2	0.77	0.83
2	146	1	3,3,2,2,2,2,2,2	0.78	0.79
2	13	1	3,3,2,2,3	0.69	0.6
2	9	1	3,3,2,3,2	0.67	0.67
2	9	1	3,3,2,3,2,2	0.67	1
2	137	1	3,3,3,2	0.74	0.84
2	122	1	3,3,3,2,2	0.75	0.8
2	115	1	3,3,3,2,2,2	0.75	0.81

2	109	1	3,3,3,2,2,2,2	0.75	0.8
2	107	1	3,3,3,2,2,2,2,2	0.76	0.83
2	15	1	3,3,3,2,3	0.6	1
2	8	1	3,3,3,2,3,2	0.75	1
2	8	1	3,3,3,2,3,2,2	0.75	1
2	113	1	3,3,3,3,2	0.77	0.78
2	101	1	3,3,3,3,2,2	0.77	0.73
2	97	1	3,3,3,3,2,2,2	0.76	0.73
2	92	1	3,3,3,3,2,2,2,2	0.76	0.74
2	8	1	3,3,3,3,2,3,2	0.75	1
2	8	1	3,3,3,3,2,3,2,2	0.75	1
2	91	1	3,3,3,3,3,2,2	0.77	0.71
2	87	1	3,3,3,3,3,2,2,2	0.77	0.7
2	8	1	3,3,3,3,3,2,3,2	0.75	1
2	81	1	3,3,3,3,3,3,2,2	0.74	0.78
7	14	8	2,2,2,2,2,0,0	0.71	0.67
7	11	8	2,2,2,2,2,2,0,0	0.64	0.67
10	18	9	2,2,2,2,2,2,0,0	0.61	1
20	9	19	3,2,1,1,1	0.67	0.8
20	8	19	3,3,2,1,1,1	0.75	0.75
20	8	19	3,3,3,2,1,1,1	0.75	0.67
34	9	33	1,1,3,2	0.67	0.75

APPENDIX III: Plant-Level Association Rules

				learning	testing
plant_1	support_p1	plant_2	pattern	conf(p2 | p1)	conf(p2 | p1)
116	37	117	3,2,2	0.32	0.32
173	22	170	3,3,2	0.27	0.27
247	31	273	1,1,1,1,1,1,2,2	0.26	0.26
247	21	273	1,1,1,1,1,2,2,2	0.33	0.33
247	31	273	1,1,1,1,1,2,2	0.26	0.26
247	21	273	1,1,1,1,2,2,2	0.33	0.33
247	31	273	1,1,1,1,2,2	0.26	0.26
247	73	273	1,1,1,1,2	0.3	0.3
247	23	273	1,1,1,2,2,2	0.3	0.3
247	77	273	1,1,1,2	0.3	0.3
247	23	273	1,1,2,2,2	0.3	0.3
247	78	273	1,1,2	0.31	0.31
247	23	273	1,2,2,2	0.3	0.3
247	68	273	2,1,1,1,1	0.25	0.25
259	22	302	0,0,1,2	0.32	0.32
259	154	302	0,0,1	0.32	0.32
259	24	302	0,1,2	0.33	0.33
356	19	259	3,3,2	0.32	0.32
356	14	259	3,3,3,2	0.43	0.43
271	255	302	0,0,1	0.26	0.26
331	31	302	2,1,0,0	0.26	0.26
331	31	302	2,1,0	0.26	0.26
302	351	335	0,0,1	0.28	0.28
335	369	302	0,0,1	0.27	0.27

CIRRICULUM VITAE

NAME: John Damon Gant

ADDRESS: 3304 Grandview Avenue
Louisville, KY 40207

DOB: Louisville, Kentucky – May 15, 1977

EDUCATION
& TRAINING: B.S., Computer Engineering and Computer Science
University of Louisville
1998-2004

MEng, Computer Engineering and Computer Science
University of Louisville
2004-2006