UNGUIDED DATA MINING

Blind Dissonance Approach

A thesis submitted to the

Division of Research and Advanced Studies
Of the University of Cincinnati

In partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE

In the Department of Electrical and Computer Engineering
Computer Science of the College of Engineering

1997

By

Cynthia M. Flowers

B. S., Allegheny College 1993

Committee Chair:

Dr. Lawrence Mazlack

Abstract

Data mining is a research area that makes use of methods from both database and artificial intelligence technologies. The goal of this project was to take the first steps in developing a new data mining method, which has been labeled The Blind Dissonance Value Algorithm. The term blind is used because the algorithm does not use any prior domain knowledge to develop the hidden patterns. The Dissonance Value refers to the main heuristic within the algorithm to limit the search space. To this end, this project included a study and comparison of current data mining techniques, a high level overview of the new algorithm, and the development of methods for testing performance. During the study, issues such as the organization within the algorithm, metrics, and classifying a successful algorithm were each dealt with in turn. With the conclusion of this study, a "good" algorithm has been defined and created as well as bringing to light many new topics for future research.

ACKNOWLEDGMENTS

I would like to thank my Thesis committee Dr. Raj Bhantagar, Dr. John Schlipf, and especially my advisor Dr. Larry Mazlack for their patience and contributions to this project.

I would also like to thank my friends who supported me throughout the entire process, especially Jennifer Seitzer and the other graduate students from the ECECS department.

Finally, I would like to thank my family for their continued support and understanding throughout the last few years.

TABLE OF CONTENTS

LIST OF FIGURES AND DIAGRAMS

Chapter 1

INTRODUCTION

1.1 Overview

Data mining has become a buzzword. To the outside world, the applications of data mining are important. Almost any type of organization can use the results and applications of data mining. It gives the large corporations the ability to make decisions based on the large amounts of data collected daily. This collected data may have lain dormant for years because of the difference in transaction and analysis technology. Often, getting information from data mining is easier than previous methods. Users are finding more applications for the results of data mining.

Insurance companies use data mining for fraud detection. The data mining algorithm searches for patterns representing anomalies.

Retail companies can use data mining for customer profiles. The data mining algorithms can go into the daily sales transaction database and develop a classification scheme based on what items were bought together.

Scientific facilities can use the data mining to analyze the large sets of data collected by satellites.

To the world dealing with the inner workings of data mining, the researcher has many challenges. A main challenge in data mining is the size of the data set being explored. Data mining offers its highest payoff when applied to very large databases. The common user of this technology will have access to this data, probably stored in a data warehouse, and will expect the algorithm to complete in reasonable time. Parallel processing and sampling methods offer two different

methods for overcoming this challenge. Another challenge is the variety of data types that occur in real world databases. The data mining algorithms must deal with all kinds of data, while maintaining the information level associated with that data. Of course, the output of the data miner cannot be ignored. It too plays a large role in the acceptance of the algorithm. These are just a few of the challenges facing research in data mining.

The term's data mining and knowledge discovery in databases (KDD) have been used synonymously in the past. KDD, as defined at the IJCAI-95 KDD Tutorial [Fayyad, Simoudis 1995] is

> *"The non-trivial process of identifying valid, novel and potentially useful, and ultimately understandable patterns in data. KDD is the overall process of detecting and preparing data, selecting projections, selecting data mining methods, extracting patterns, evaluating patterns as potential "knowledge," and consolidating knowledge."*

From this definition, data mining is a step in the overall process of KDD. In this paper, data mining refers to a class of methods for extracting patterns from data. Data mining seeks to discover noteworthy, unrecognized associations between data items in an existing database. One definition of data mining is the non-trivial extraction of interesting high level patterns from data.

1.2 A statement of purpose

This paper is about exploration of data mining and the first stages of algorithmic development for a new method for data mining. In the development of the algorithm, the focus was simplicity. A first question formulated for this study was the following: could data mining be performed using no domain knowledge and very simple calculations? This led to further questions involving if a possible

limit of domain knowledge required could be discovered, if a different heuristic could be applied and still maintain a high level of information within the result.

The focus of this study was on the use of domain knowledge and heuristics. The approach involved eliminating domain knowledge because it may bias the results and limit what may be discovered. A simple informational heuristic was chosen because of its computational simplicity. The heuristic is based on the progressive reduction of cognitive dissonance (dissonance is how much disorder or chaos). The heuristic assumption is that reducing cognitive dissonance increases information. If successful, the results of the algorithm under development would be simple classifications or descriptions of the database.

The result of this study was the development of the first version of the Blind Dissonance Value algorithm for data mining. The study can be broken into the following areas:

- A background study of data mining techniques and KDD systems
- The initial high level designs of the algorithm.
- The development of a testing and experimentation scheme.
- An implementation of the high level design.
- The beginnings of algorithmic refinement.

Throughout the entire study, many questions and options have been developed.

Chapter 2

LITERATURE COMPARISON

From the research conducted for this study and concurrent studies by my adviser, we have developed a further classification system for the current data mining algorithms. This classification is based on the amount of domain knowledge used by the algorithm to improve both performance and the results from data mining. Algorithms where domain knowledge is required and used for concept hierarchies and generalizations plus domain knowledge coming from the user's active participation in the execution of the algorithm have been classified as "Guided" methods. When no domain knowledge is involved, the algorithm is classified as "Unguided."

In addition, data mining algorithms can be classified according to their methods of processing the actual data. In our classification, algorithms can fall into two categories: incremental or non-incremental. Incremental algorithms discover information by processing individual records within the data set one at a time. Non-incremental algorithms use the entire data set as a whole to discover information. These methods play a role in the issue and techniques of up keeping the discovered knowledge. Data can either be changed or added to the existing data set. Incremental and non-incremental algorithms approach this problem differently.

2.1 Domain Knowledge

A common issue within data mining algorithms is the use of domain knowledge. Domain knowledge is knowledge about the specific domain characterized by the database as a whole rather than information about the data itself, such as field width or attribute values. Domain knowledge is commonly used within the algorithm to improve efficiency and the relations discovered.

The need for domain knowledge in machine learning comes from the idea of bias. Any incomplete collection of data can be explained to the same degree by several different theories. Selecting any one of them requires some criterion other than goodness of fit [Quinlan 1986]. The issue involves the extent to which data mining programs should employ special knowledge of the domain in which they are attempting to learn. Information about the data and how it was assembled can be used to help the mining process. However, providing knowledge about the domain is a different story. Many believe that algorithms must exploit what is known about a domain to make significant discoveries. This is guided discovery and these systems are commonly looking for only the most interesting result. However, domain knowledge may limit objectivity. We see what we want to see. Those believing that domain knowledge limits or creates a bias about what is discovered are classified as supporting unguided discovery. Unguided discovery tools attempt to find many possible relations from the database.

2.2 Incremental Characteristics of Databases

A database is characterized by growth. Transactions and updates are everyday occurrences and can result in rapid changes within the database. Therefore, the

validity of the discovered information must be taken into account as the information within the database changes. An ideal data mining algorithm could modify what has already been learned considering new information at hand.

An algorithm is classified as incremental if it can take what has been learned from the previous data mining sessions and modify it to adapt to the new information. This new information is defined as the occurrence of new records added to the data set. It does not attempt to incorporate changes that occur within existing data, such as deletions or corrections, to the data mining scheme. Incremental algorithms can process the new information as a separate instance. They do not have to reevaluate every piece of data again but still allow the search space to grow while maintaining their original performance level. An advantage of incremental methods is their tendency to closely integrate learning and performance, which seem to be more appropriate for modeling human behavior and in autonomous agents [Langley 1996].

However, occasionally the results of incremental systems are dependent on the ordering of the instances presented to the system [Fisher 1987]. A non-incremental algorithm develops classification schemes based on the total number of instances available to the learning algorithm. One advantage of non-incremental methods is that they can collect statistics about all training cases, making more information available to make decisions in later sessions. However, to incorporate new information within the data set, non-incremental data mining must be done again on the entire set of data.

2.3 Comparison of Algorithms and Systems Common in Data Mining

Authors [Year]	Discovery Method	Modification
Anwar, Beck [1992]	Guided	Incremental
Dhar, Tuzhilin [1993]	Guided	Non-incremental
Clark, Niblett [1989]	Guided	Non-incremental
Drastal, Czako, Raatz [1992]	Guided	Non-incremental
Dzeroski, Lavrac [1993]	Guided	Non-incremental
Han, Cai, Cercone [1992]	Guided	Non-incremental
Hayes-Roth, McDermott [1978]	Guided	Incremental
Pazzani, Kibler [1992]	Guided	Non-incremental
Piatetsky-Shapiro, Matheus [1992]	Guided	Non-incremental
Quinlan [1986]	Guided	Non-incremental
Yoon, Kershberg [1993]	Guided	Non-incremental
Zytkow, Zembowicz [1993]	Guided	Non-incremental
Zhong, Oshuga [1993]	Guided (both)	Non-incremental
Zytkow, Baker [1991]	Guided	Non-incremental
Agrawal, Imielinski, Swami [1993]	Unguided	Non-incremental
Chiu, Wong, Cheung [1991]	Unguided	Incremental
Fisher [1987]	Unguided	Incremental
Ragaven, Rendell, Shaw, Tessmer [1993]	Unguided	Non-incremental
Shen [1991]	Unguided	Non-incremental
Smyth, Goodman [1992]	Unguided	Incremental
Ziarko, Shan [1995]	Unguided	Non-incremental
Ziarko [1991]	Unguided	Non-incremental

Table 2.1 Comparison of Data Mining and Relevant Machine Learning Algorithms

Table 2.1 offers a summary of a comparison of popular data mining algorithms and systems. The Method of Discovery category represents the use of domain knowledge. Its value will be either guided or unguided. The Modification column represents the ability to incorporate newly discovered information and uphold the validity of the discovered knowledge. Its value will be either incremental or non-incremental. In many systems, both guided and unguided discovery techniques were used. For example, Zhong and Ohsuga implement a method of unguided discovery when user input is not available [Zhong, Ohsuga

1993]. However, it offers the user this choice and is therefore categorized as Guided (both).

2.3.1 Guided Discovery Methods

From Table 2.1, many guided systems encode domain knowledge directly into the data mining algorithm. Han, Cercone, Cai [1992]; Dhar, Tuzhilin [1993]; and Drastal, Czako, Raatz [1992] use domain knowledge in this way. Domain knowledge can be encoded within data dictionaries as in the Abstract Driven Discovery of Dhar and Tuzhilin [1993]. In addition, it can be encoded in a concept hierarchy set that is used for generalization, as in Han's DBLearn [1992]. The data dictionaries of Dhar and Tuzhilin are characterized with user-defined generalizations that allow different data types to be generalized. Concept hierarchies within DBLearn are used to generalize attribute-values of a specific data type. When the knowledge is encoded, the systems usually maintain a high level of efficiency, but are limited to being used within that domain.

Another way domain knowledge is used in the data mining processes is through user input throughout execution (Piatetsky-Shapiro, Matheus [1992], Anwar [1992]). Domain knowledge coming from user input is commonly used to guide discovery as in Zytkow and Baker's FORTY-NINER system [1992] and the Knowledge Discovery Workbench system of Piatetsky-Shapiro and Matheus [1992]. User input is accepted by FORTY-NINER that aids in refining discovered regularities. The user can select the most promising attributes or regularities to continue the refinement process. The SPROUTER system [Hayes-Roth 1978] also allows the user to limit the search space by defining relevant attributes. The user plays a similar role, that of evaluating potential relations, in the Knowledge Discovery Workbench.

The decomposition-based induction algorithm of Zhong and Ohsuga [1993] applies user input in another way. Rather than allowing the user to study each potential relation individually, the user is asked to set a threshold value to be used within probability distributions. This threshold value can correspond to a special attribute at the beginning of the search.

2.3.2 Unguided Discovery Methods

Unguided methods of discovery use heuristics rather than domain knowledge to discover interesting information. The knowledge supplied to these systems only includes the characteristics of the database. Therefore, these systems can be applied to more than one domain area. The efficiency of these methods is still being improved. These heuristics are from three main areas: rough set methods, statistical methods, and information theory. Rough set methods are shown by the work of Ziarko [1991,1995]. Statistical methods can be shown in the work of Shen [1991].

Heuristics based on information theory are found in the work of Agrawal, Imielinski, and Swami [1993], and Smyth and Goodman [1992]. Agrawal, Imielinski, and Swami incorporate a heuristic (1) that is based on Quinlan's information gain ratio [1986]. I is the information content of the attribute A_i, where n is an object in the database.

$$(1)\quad I(A_i) = -\sum_j \frac{n_{ij}}{n} \log \frac{n_{ij}}{n}$$

Smyth and Goodman, on the other hand, create what they call a J-measure [Smyth, Goodman 1992] (2). Where j is defined as (3), where X and Y are two attributes, y is a value within attribute Y and x is a value within attribute X. It

is the average information content of a rule. This measure can be viewed as a special case of the cross-entropy measure by Shore and Johnson or the discrimination measure (see [Pal 1992] for further information) that defines the information theoretic similarity between two probability distributions.

$$(2)\quad J(X{:}Y=y) = p(y) * j(X{:}Y=y)$$

$$(3)\quad j(X{:}Y=y) = p(x|y)\log\frac{p(x|y)}{p(x)} + \frac{(1-p(x|y)\log(1-p(x|y))}{1-p(x)}$$

2.3.3 Incremental Design

Those algorithms meeting the incremental definition have expected their databases to be constantly changing, where each transaction could play a large role in the meaning or content of the database. When the content of the database changes, the descriptions and relations found through data mining could become invalid against the new database. In incremental algorithms running the data mining algorithm from scratch is not necessary because the new information can be entered into the system as a new instance. These new instances will result in altering the current data mining results rather than rediscovering all previous relations. Fisher's COBWEB algorithm [1987] is the defining algorithm of the incremental design.

One guided discovery method within Table 2.1 considered to be incremental is the SPROUTER algorithm by Hayes-Roth and McDermott [1987]. This algorithm is a machine learning algorithm for inducing abstractions but is relevant to data mining. SPROUTER has been classified as incremental because abstractions are

developed one relation at a time, rather than developed from the entire table of relations. In this way, if a new abstraction occurs throughout the algorithm, it is saved and the system increases the number of abstractions found. Another example is found in the ITRULE algorithm of Smyth and Goodman [1992]. ITRULE uses unguided discovery and incremental methods. Calculations are done based on information theory on individual instances, allowing new instances to be incorporated into the learning process anytime.

2.3.4 Non-incremental Design

Many algorithms within Table 2.1 do not place any emphasis on the characteristic growth of the database. These algorithms commonly implement a form of guided discovery. They require the use of all data instances when doing data mining. Occasionally, these algorithms have reached a level of efficiency where rerunning the data mining algorithm when the database change is reasonable. Another reason may be the nature of the information that is being mined. If the data is constantly changing or is collected on a daily basis (historical trending), the data mining algorithm may have to be rerun over the data set to adapt to the new information. The defining algorithm within table 2.1 of this type of discovery is Quinlan's ID3 algorithm [1986].

Chapter 3

BACKGROUND

Data mining is an application area of machine learning. It is appropriate, then, that we start our background discussion with some topics from machine learning. By expanding on certain machine learning algorithms, primarily those under the category of learning from examples (or observations), we can identify the influence of machine learning on data mining. Data mining has become such a popular area that the term is commonly misused. Machine learning and data mining, while very similar, differ in the following ways:

Data mining is concerned with finding "understandable" knowledge, while machine learning is concerned with improving performance of an agent.

Data mining is concerned with very large, real-world databases, while machine learning is typically, but not always, looking at smaller data sets. Therefore, efficiency questions are much more important for data mining.

Machine learning is a broader field that includes many different paradigms of learning. (Piatetsky-Shapiro, 1995, http://www.gte.com/kdd/)

3.1 Machine Learning

Machine learning involves building computer programs that construct or improve knowledge. Learning can be accomplished in a variety of ways. It is common to break down machine learning into five main paradigms: neural networks, case-based learning, genetic algorithms, rule induction (learning from example), and analytical learning. These paradigms differ in basic ideology and methodology in most cases, but all are focusing on the goal of improving knowledge.

The search space of any machine learning algorithm plays a central role in the way that algorithm performs. Since the search space defines the set of all concepts the system may learn, the structure and manipulation of this set is very important. All algorithms attempt to avoid a combinatorial explosion of this search space. Domain knowledge, heuristics, and analogies have been used successfully in many systems to restrict the search space and avoid combinatorial explosion.

When searching for classification schemes or descriptions of the search space (clustering methods), the number of different ways to partition the search space is an important consideration. The number of partitions of this search space can lead to combinatorial explosions. A general approach to calculating the number of ways to partition N elements in m subsets is:

$$P(N,m) = \frac{1}{m!}\sum_{j} (m,j) - 1^{j}(m-j)^{N}$$

This calculation assumes that the *m* subsets are distinct non-empty sets where in the partitioning of the *N* objects, the order of the *m* subsets is irrelevant. *P (N, m)* is a function that grows exponentially fast in *N* [Duran, Odell 1974].

3.1.1 Heuristic Search

Machine learning has dealt with the problem of combinatorial explosion by using heuristics within their search techniques. Heuristics attempt to estimate either the relative benefit or the relative cost of continuing the search along a path using a certain function. These functions map problem state descriptions to levels of desirability that are usually represented as numbers. The purpose of a heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow when more than one is available. It is, therefore, important that efficient mapping is developed from the problem space to the measurement scheme used by the heuristic function.

While heuristics can effectively eliminate combinatorial explosion, there is also a possibility that the heuristic function could eliminate potential conclusions that may be relevant. Heuristics are commonly found in tree searches. An example of one heuristic search is the alpha-beta search. In alpha-beta, the search space is limited based on a function of the maximum and minimum cost values at each level of the tree. Depending on the combination of these values, sub-trees are eliminated based on the assumption that, because they do not meet the function requirements, they do not lead to the best conclusions.

3.1.2 Decision Trees

Quinlan's ID3 algorithm [1986] is the definitive work on decision trees. ID3 [Quinlan 1986] is a non-incremental classifier system. A guided learning system constructs decision trees from a set of examples. Domain knowledge is encoded into the example instances as class information. These examples are represented by attribute value pairs and are very similar to database tables. The algorithm assumes that each example in the training set belongs to a mutually exclusive class.

ID3 [Quinlan 1986] uses a top down irrevocable strategy that limits the search space as the algorithm executes. ID3 [Quinlan 1986] builds a tree structure that can be used to classify the examples in the training set. Any correct decision tree for the training set will classify objects in the same proportion as their representation in original training set.

To construct these trees optimally, ID3 [Quinlan 1986] uses an information theoretical heuristic based on Shannon's entropy measures to select the attributes to place into the tree. To explain the heuristic, we define two classes: *P* containing all positive examples and N containing all negative examples. Let the set of examples *S* contains *p* elements of class *P* and *n* elements of class *N*. In information theory, a measure is defined for how much information is needed to decide if an arbitrary example in *S* belongs to *P* or *N*:

$$I(p,n) = -\frac{p}{(p+n)}\log_2\frac{p}{(p+n)} - \frac{n}{(p+n)}\log_2\frac{n}{(p+n)}$$

If attribute A is used as the root that will partition S into sets $\{S_0...S_v\}$, the information needed to determine if an element in S_i belongs to P or N is $I(p_i, n_i)$. This heuristic selects the attribute providing the highest information gain, which is the attribute that reduces the information needed in the resulting sub-trees to classify the elements.

The use of a heuristic based on information theory has since been used in many algorithms and studies concerning machine learning as well as data mining. These include work by Agrawal, Imielinski, and Swami [1993]; Han, Cercone, and Cai [1992]; Smyth, and Goodman [1992]. Using the decision tree method, the resulting classification schemes have been more coherent than classifications based only on classical statistical methods.

3.1.3 Conceptual Clustering

Conceptual clustering is a machine learning technique that applies an unsupervised (unguided) search. In unguided search techniques, the aim of the learning process is to discover regularities in data. Usually, no domain knowledge is made directly available to the algorithm. Instead, these systems must depend on heuristic techniques to limit the search space. Since these heuristics are not limited by domain knowledge that may be "hard coded" into the heuristic, these methods can be reapplied in a variety of domains.

The COBWEB system, developed by Fisher [1987], is an example of an unguided clustering algorithm. It is similar to ID3 [Quinlan 1986] in that it also uses

information theoretical heuristics to decide class membership. However, it differs from ID3 in two ways:

COBWEB is an unguided/unsupervised learning technique. The example instances used as inputs to the system do not contain class type information.

COBWEB is an incremental system. The classification scheme can incorporate new examples as they occur to the system. COBWEB offers an alternative method to the decision trees of ID3 where domain knowledge is not needed to perform classification.

The heuristic implemented within COBWEB (category utility) [Fisher 1987] is based on the predictability of the produced classification structures. It favors clustering that maximizes the potential for inferring information. In doing this, it attempts to maximize the similarity within the class and interclass differences. Fisher's category utility is the increase in the expected number of attribute values that can be guessed given a set of n categories, over the expected number of correct guess without such knowledge.

$$CU(C_1, C_2, \ldots C_n) = \frac{\sum_{k=1}^{n} P(C_k) \sum_i \sum_j P(A_i = V_{ij} | C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2}{n}$$

Where $A=V_{ij}$ is any attribute pair, C_k is any class, k = all classes, and n = number of categories.

3.1.4 Measuring Uncertainty

In machine learning, every system must deal with uncertainty. There are two kinds of uncertainty, fuzzy and non-fuzzy. Fuzzy uncertainty deals with situations where boundaries of sets are not sharply defined. Non-fuzzy

uncertainty occurs when there are clear set boundaries, but uncertainty in what elements belong within each set. This project deals with non-fuzzy uncertainty.

Uncertainty can be dealt with by creating measurements and heuristics based on entropy values. By measuring the amount of dissonance, confusion, or chaos, the system can limit the search space by attempting to limit these values. For example, two aspects of non-fuzzy uncertainty are "nonspecificity" and conflict (confusion/discord) [Pal 1992]. Yager [in Pal 1992] proposed a measure of dissonance (conflict) in a body of evidence. This measurement is based on the idea that a conflict exists whenever the evidence suggests that the element of concern may belong to two or more disjoint subsets.

$$E(m) = -\sum_{A \in F} m(A) \log Pl(A)$$

In another example, Hohle [in Pal 1992] measures confusion as shown below. *C* represents the conflict that arises when two evidential claims *m (A)* and *m (B)* conflict within the same body of evidence and when $B \notin A$.

$$C(m) = \sum_{A \in F} m(A) \log Bel(A)$$

For a larger discussion on uncertainty measures based on entropy measurements see [Pal 1992]. Also, Shannon's entropy paper is a large source for many algorithms, including Quinlan's measurements found in ID3 [1986].

3.2 Database Background

In a database, data is organized into tables of fixed-length records. Each record is then made up of a fixed set of fields, one value for each field. Individual

fields within the database cannot contain multiple pieces of data. A separate table, known as a data dictionary, contains information about each field including its name, type of data, and possibly, a range of permitted values.

A common characteristic in many database management systems is the ability of the system to represent data using an abstract model. Through this model, the user can gain information in understandable combinations rather than raw pieces of data. The models used within DBMS are usually based on the real world, where all information concerning the world is in the table. When applied to machine learning, each record represents an example that can be used for learning purposes. Tables organized in this fashion can be called attribute-value tables. In viewing the table in this manner, existing machine learning techniques based on attribute-value data structures can be carried out.

3.3 Data Mining

Data mining involves methods for finding new descriptions and patterns within data. Often the data structures involved receive the most attention during design. The following are common characteristics within databases that play an important role in designing data mining techniques.

Dynamic Data - A common characteristic among databases is the changing data within the system. The ability of managing these transactions and maintaining the validity of this ever-changing data is a strong point of databases. Data mining techniques must keep this property of databases in mind because of the effect it has on the validity of the discovered knowledge. If data mining is set up as a non-incremental system, and the information within the database is affected by a few transactions, the mining algorithm must be repeated on the entire database each time the database is altered. However, if an incremental system is developed, the new transactions can be incorporated into the mining algorithm. If the knowledge discovered is to be accurate, and if the data changes at any point through the system, the newly discovered knowledge must be updated to reflect the database [Zhong, Ohsuga 1993].

Noise and Uncertainty - Noise, also known as incorrect data, can be a large concern in real-world databases because wrong values could be entered for one reason or another. The noise leads to uncertainty within the data. Noise has been a common topic in machine learning and has been dealt within both statistical methods and conceptual methods [Mango, Kodratoff 1987].

Incomplete Data - Data can be incomplete either through the absence of values in individual record fields or through the complete absence of data fields necessary for discoveries. This is common in databases, since each record in a table must have the same fields, regardless if the information exists or not. Incomplete data has also been studied within the area of machine learning [Quinlan 1986]. Some systems solve this problem by attempting to figure out unknown values at run time [Langley 1981], while others simply ignore the record containing incomplete data.

Sparse Data - Information in a database is often sparse regarding the density of actual data records over the potential instance space. For example, rare cases may exist, but may contain important information that could be used as delimiters or exceptions. If the number of original instances covered by developed pattern is small, these exceptions or delimiters may be neglected or may appear to be more important than they really are. This leads to the development of a measure to keep track of how many instances fall within the given pattern. These measures can give the user a better idea of the concepts within the database including how to use them. DBLearn simply keeps track of the number of instances represented by a given summary record [Han 1992].

Data Volume - The large amount of data contained in these databases is constantly growing rapidly. The volume itself is beginning to dictate the types of operations that can be done in data mining. The volume of data must be controlled to search efficiently for new knowledge.

Within data mining, uncertainty must be measured to form correct relations and deal with noisy, sparse, and incomplete data. Quinlan's information theoretical

heuristic [1986] is commonly used in these systems: DBLearn [Han 1992], abstract driven discovery [Dhar, Tuzhilin 1993], the knowledge discovery workbench [Matheus 1993], and the system developed by Agrawal, Imielinski, Swami based on information theory [1993]. Those systems attempting to use other methods rather than information theory employ statistical methods [Zytkow, Baker 1991], probability distributions [Zhong, Ohsuga 1993] and rough sets [Ziarko 1991].

3.4 Guided Data Mining Techniques

The goal of all data mining techniques is to produce the most interesting results for the user. To reach this goal, several methods have developed ways to use domain knowledge to maximize the level of interest of the results. In these systems, domain knowledge can be encoded within the algorithm to allow for generalizations of data, (to shrink the search size), or by allowing the user to determine what input could produce the most interesting answers. These systems have the advantage that they are usually efficient and produce results that are of interest to the user. The following sections describe individual algorithms and popular systems in the data mining community that relate to this project.

3.4.1 DBLearn - Attribute-Oriented Induction

DBLearn [Han 1992] is a system that uses domain knowledge to generate descriptions for predefined subsets in a relational database. The starting set of instances for this algorithm is the entire table. The aim is to generalize this table to a class description, a much smaller table covering all examples belonging to this class. The key to this approach is to move up the concept tree using

attributes to gain generalization. When carried out, it applies well-developed, set-oriented database operations and substantially reduces the computational complexity of the database learning process.

In DBLearn, three primitives are provided for learning in databases: task relevant data, background knowledge, and the expected representations of learning results. Allowing the user to state the database fields that could be important for the results collects task relevant data. Background knowledge is contained in a concept hierarchy table that controls the generalization process. These hierarchies are organized from general to specific, the most general cases being "any" and the most specific concepts corresponding to the attribute values within the original database. Finally, the expected representations of these learning results are defaulted to take the form of logical rules.

DBLearn uses two generalization operations to limit the search space and create a class description: attribute removal and moving up the concept tree. Attribute removal drops all attribute-value pairs for a particular attribute in the table, eliminates a conjunct, and thus generalizes the record (Projection in relational algebra). Attribute removal occurs when there are multiple values for an attribute, but no higher concept is provided in the concept hierarchy. Since these values cannot be generalized, they are dropped. Upward movement through the concept tree occurs when values in a particular attribute are replaced with a more general value in the hierarchy, thus generalizing the relation. As the search space becomes smaller and more generalized, the distributions of records from the original table are propagated by use of a "Vote" field within the current table. This field keeps track of how many instances can

be covered by the individual generalization. This knowledge can be used in determining how the results are applied.

Overall, DBLearn is controlled by a threshold value. The algorithm continues to generalize the given table until a threshold value, usually set by the user, is either met or exceeded. Threshold values relate to the number of records within a final table. They are used to control the complexity of the final logical formula used to describe the database. In addition, threshold values are also used to control generalization. A low threshold value may result in over generalization of attribute values. A trade off exists between the number of generalizations done on the data set and the complexity of the resulting logical formula.

DBLearn is highly dependent on domain knowledge. While certain types of attributes can be generalized using statistical methods (mainly attributes with scalar values), the algorithm is dependent on the quality of the concept hierarchies provided. Assuming the background knowledge is available and because attributes usually have a finite number of unique values, these hierarchies can be created feasibly. Regardless of these limitations, this dependence is effective in limiting the search space while producing correct results.

3.4.2 FORTY-NINER System

The FORTY-NINER (49er) system developed by Zytkow and Baker [1992] uses a combination of techniques to discover new information. This system searches for regularities, defined as limitations on the set of all possible facts, from within

the database. Regularity exists if some events are either impossible or less probable than others are. It is based on the machine discovery techniques used in bacon and FAHRENHEIT adjusted to meet the specific needs of database exploration. It is a non-incremental system allowing user interaction to guide discovery in intermediate stages. It operates in two phases. In phase one, 49er does a search for two-dimensional regularities based on reversing the method found in the FAHRENHEIT for finding regularities. In the second phase, the regularities are generalized to include more dimensions. These two phases can be repeated many times until the user is satisfied.

In finding regularities, three operations are done on each attribute: aggregation, slicing, and projection [Zytkow, Baker 1991]. Aggregation combines values of an attribute into classes of abstraction. This is done to reduce the number of values within the search. 49er will aggregate attribute values into two classes: lower and higher. Slicing, also known as data partitioning, selects all elements of the data set that have a value v_i in attribute A_i to narrow the range of data. Slicing is only done on aggregated values because it is possible that stronger regularities could exist in one or more slices of A_i. Projecting the attribute A_i is equivalent to ignoring this attribute. All records are included regardless of the value of A_i. When used with aggregation, it is helpful in reducing the sparseness of data in the entire set.

A partitioning search is used to produce subsets of the database to be searched for two dimensional regularities. Partitioning generates all combinations of the given independent and dependent variables using the basic operations described above. Partitioning is controlled by the selection of the sets of independent and

dependent variables and by the minimum size of a data set that will not be further sliced. Within each data subset, after all independent variables have been sliced or projected, the find-regularity algorithm attempts to find relationships between a dependent attribute *D* and an independent attribute *I*. Regularities are computed using statistical methods. Each one is saved [Zytkow, Baker 1991].

3.4.3 Abstract Driven Discovery

The system developed by Dhar and Tuzhilin [1993] is a pattern discovery process. This system uses domain knowledge not only to limit the search space, but also to bias the search by allowing the user to easily comprehend the patterns. This use of domain knowledge attempts to summarize the original data in terms familiar to the user, allowing the discovered patterns to be easily comprehended by the user. The system is based on the idea of abstraction of information: attribute values in an abstracted database can also be predicates or views of the original database maintaining the same informational value. This system develops a summary table with the aid of the domain knowledge, and then uses predefined data mining techniques similar to those found in DBLearn [Han 1992] and statistical methods on the summarized table.

To summarize the data, domain knowledge is encoded in a data dictionary. The data dictionary defines terminology according to the database schema. It incorporates the following: the vocabulary containing a set of the user defined predicates or views, a classification hierarchy, and a set of abstract functions. The predicates are used to translate/summarize the raw data into an easily understood form. The classification hierarchy is similar to the concept hierarchy

found in DBLearn [Han 1992]. Finally, the abstraction function of an attribute maps the domain values of the attribute into another domain.

3.4.4 Knowledge Discovery Workbench

The knowledge discovery workbench (KDW) is a system for knowledge discovery based on the idea that the best results can be found by using a combination of different algorithms. It has been designed as an interactive workstation for data exploration. It includes a graphical user interface for ease of use. By setting up an interactive system, the discovery tasks can use information from the user to limit the exploration space. It can develop knowledge that is both relevant and of interest to the user. KDW incorporates both statistical and clustering methods to perform data mining. Clustering is performed using a linear clustering algorithm to detect multiple linear groupings of data points. Data summarization is the process of deriving a characteristic summary of a data subset that is interesting regarding domain knowledge and the full data file. This is done in KDW using two by two contingency tables and other statistical techniques. Finally, to discover characteristics that can be used to classify data points into a class, KDW constructs a decision tree based on ID3 [Quinlan 1986] and Frawley's Function-based Induction [Frawley 1991]. In addition, future work is being done to include discovery of anomalies using logistic regression and discovery of changes within the data using statistical methods.

3.4.5 Decomposition-based Induction

Decomposition-based induction [Zhong, Ohsuga 1993] discovers conceptual clusters from databases. It is based on the concept of Simon and Ando's near-complete decomposable [see Zhong, Ohsuga 1993 for reference] commonly used

in economic theory. The important feature of this approach is the formation of conceptual clusters or sub-databases through analysis and deletion of noisy data when decomposing a database. The decomposition of databases is also a kind of aggregation and abstraction.

In this method, knowledge discovery is accomplished using three types of database spaces: the instance (an ordinary database set), the probability space, and the learning space. Probability space is used to store distributions, created using a probability distribution matrix, describing the status of the data. The PDM is created based on the probability distributions of dependency relations between any two attributes because of the classification knowledge inherent in the dependency relationships. The probability distributions are calculated as follows:

Let a = $\{a_1, a_2, \ldots a_n\}$ and b = $\{b_1, b_2, \ldots b_m\}$ be the sets of two different values of any two attributes in the instance space. Using conditional probability

$$P(x_i \mid x_j) = \frac{x_i \cap x_j}{p(x_j)}$$

From this, p_{ij}, the probability distribution is defined as

$$p = \frac{p(x_i \mid x_j)}{N}$$

Where N is the number of attributes [Zhong, Ohsuga 1993].

The elements making up the diagonal in the matrix are specified by a threshold value that the user sets. The elements larger than the threshold values are moved into the diagonal sub-matrices and the elements smaller than the threshold values are moved outside the diagonal. The user to correspond to a special attribute, or be determined by finding the optimum decomposition, which is an unguided method, can either specify this diagonal. Irrelevant elements are defined in terms of the diagonal formed within the matrix.

The learning space stores the probability distributions and errors for learning. It is used to control when the decomposition of the probability/instance space must be re-computed, maintaining the discovered knowledge throughout a dynamic database.

3.5 Unguided Data Mining

Unguided data mining can be viewed as a "fishing expedition." The algorithms are designed to produce multiple results in hopes that the user will find at least one that he or she deems new and interesting. To find interesting results without the use of domain knowledge, other methodologies have been implemented: rough sets [Ziarko 1991], [Ziarko, Shan 1995], methods based on information theory [Smyth, Goodman 1992], and statistical methods [Shen 1991]. Unguided algorithms offer the user unbiased results, which may produce relations that previously had been overlooked.

3.5.1 Rough Sets in Data Mining

Pawlak [1981] developed the theory of rough sets in the 1980's. It is a form of analysis complementary to statistical methods of inference, providing new insights into the structural properties of data. The primary goal of rough sets is the discovery, representation and analysis of data regularities. The focus of this technique is on the investigation of structural relationships in data rather than probability distributions. These methods are very useful for reasoning from qualitative or imprecise data.

Ziarko [1991] has set up a system known as ROUGH that is based on rough set theory. Using rough set measurements and techniques, lower approximation, boundary regions, and upper approximation, ROUGH measures dependencies among attributes. Along with finding these dependencies, ROUGH also can find the best minimal set: the minimum set of attributes maintaining the discovered information. Results from this system are stored in decision table format and are transformed into if-then production rules using an algorithm also based on rough sets.

More recently, Ziarko and Shan [1995] have developed an algorithm to search for domain classifications. This algorithm is also based on rough set theory and is made up of two stages: cluster generation and search for classifications. In the first stage, each cluster formed contains attributes that are directly or indirectly dependent upon each other. In doing this, the search space is limited by avoiding independent attributes. The second stage is the computation of all cluster subsets that are of limited complexity and contain a minimum number of elements within their classes to allow for credible estimations of probabilities.

The algorithm used for computation of subsets starts with an empty subset of attributes and gradually expands the generated subset into a tree structure until the end criterion is reached.

3.5.2 Unguided Mining: Heuristics Based on Information Value

Agrawal, Imielinski, and Swami [1993] have developed an algorithm based on the idea that many types of information to be discovered through data mining can be produced using combinations of basic operations. The basic structure used in these operations is the concept of a string. A string here is an ordered sequence of a (method, value) pair. A value can be atomic or an interval in case of methods returning values that can be ordered. These basic operations include: (a) generate and measure new strings, (b) combine strings, (c) filtering strings, and (d) select the best string to be used in the final rule. Step (c), the filtering step, retains strings corresponding to the attribute that maximizes Quinlan's information gain ratio [1986].

The selection of the best string is the group with the largest frequency. Its strength is based on the ratio of the winner group frequency to the total frequency for the string across the database that is above a precision threshold. The precision threshold is an adaptive function of the length of the string. It is used to distinguish weak and strong results. The system reports only strong results.

3.5.3 Statistical Methods in Unguided Mining

Knowledge can be discovered from large knowledge databases as regularities. Shen [1991] uses induction to generate hypothesis and statistical methods to

evaluate the hypothesis to discover a limited number of regularities. Using a weak method of generate-and-test, knowledge guides the data collection and hypothesis evaluation. Hypotheses generated are evaluated using both prior knowledge and statistical techniques. The knowledge used is based on set manipulation and syntactical differences within the database. The knowledge used within this algorithm is generated during the algorithm. No outside domain knowledge is used.

3.6 Background Summary

As seen through the research, much work has been done in the guided discovery technique using both domain knowledge and user interaction. There is still much to be done with systems using more of an unguided method - limiting both the domain knowledge and the amount of user interaction.

In the future, some individuals believe that more systems using a variety of methods to discover new relationships within databases will become popular. However, in order for these to be successful, the algorithms to be used must first be developed, refined, and made to scale.

Chapter 4

BDV ALGORITHM

This project will focus on the development of an algorithm for data mining using a blind dissonance value approach (BDV Algorithm). It is called blind because the algorithm will not be provided with background knowledge that could be used to limit the search space. It will be limited to nominal data values containing almost no noise. This is the first step in the development process of the algorithm. The algorithm will be improved and revised in future research. The project determines the best methods within the technique to accomplish data mining using blind dissonance values.

The approach for data mining in this project will be based on the use of dissonance values within attributes. These dissonance values are founded on principles of information theory. They will measure disorder within an attribute. Low dissonance values will correspond to high coherence present within the table. Those attributes that maintain a low dissonance value will be considered to contain more information.

Data mining based on dissonance values is a general technique that can be done in a variety of ways. The best method is an open question. For example, data mining could employ clustering methods within the data mining system. Additionally, some systems separate the tasks of data mining into first finding dependencies within the data, and then using them to limit the search [Piatetsky-

Shapiro, Matheus 1992]. When the database contains continuous scalar values, the algorithm may carry out some form of analysis on the data before doing mining to limit the search space. This analysis could be based on techniques from the areas of rough sets, statistics or fuzzy. On the other hand, the algorithm could use domain knowledge to generalize the attribute value pairs to limit the search space. In addition, many different ways to handle noise and boundary values within a database exist. These could lead to different ways of mining the data.

4.1 Goals

The goal of data mining is to identify a set of interesting patterns. To attain this goal, two main tasks exist: (a) recognizing a pattern within the database and (b) determining if the recognized pattern is interesting. This study was concerned with the first of these tasks.

The task of recognizing interestingness is a computational difficult task in itself. The difficulty is because of its dependence on domain knowledge as well as the difficulty in embedding this type of knowledge into a context sensitive structure [Lenat, Guha 1990]. Therefore, recognizing interestingness has been considered a separate and possibly future research topic because of the size of the problem.

The BDV algorithm developed within this study focused on finding relationships using unguided search strategy. The main emphasis of this project was to determine if applying simple machine learning techniques and information theoretical heuristics could recognize the information within the database. The

underlying question was the extent of useful information that could be discovered using simple techniques with a minimum of domain knowledge.

The study considered if data mining could be accomplished through a progressive reduction of cognitive dissonance. The underlying hypothesis was that tables maintaining attributes with reduced dissonance have increased information value and thus may reveal interesting relationships. The BDV algorithm assumes that reducing cognitive dissonance will increase information value and in turn capture interesting knowledge.

4.2 An Overview of the Algorithm

The study conducted on the BDV algorithm focused on three main areas within the algorithm called partitioning, aggregation, and terminating requirements. Partitioning is a main action taken within the algorithm. It can be defined as creating new tables (partitions) based on ideas found in simple SQL select statements according to the attribute defined by the dissonance values and heuristics involved.

```
SELECT*
FROM Partition1
WHERE attribute A = value 1.
```

Listing 4.2.1 Example SQL statement for partitioning

Aggregation is another action the algorithm takes. In this algorithm, aggregation is used to refer to the "generalization" or abstraction of attribute-values in hopes of further reducing the search space. Because this algorithm is "blind," aggregation methods are based on abstraction sets collected throughout the algorithm. Abstraction sets are BDV's answer to concept hierarchies or dependency lists. Whenever an attribute is removed based on containing a high

level of coherence, the algorithm collects all possible dependencies that may exist within the current table/partition. These are then used in the future to abstract/replace the values in attributes containing a high level of dissonance. Finally, terminating requirements are implemented to ensure that the algorithm stops with a result that may prove interesting to the user.

4.2.1 Partitioning

The BDV algorithm attempts to limit search space by partitioning the original table of examples into smaller tables based on a reduction of dissonance within attributes. The original search space is made of the data set represented within a flat file. Partitioning the search space according to cognitive dissonance within the partitions (tables resulting from partitioning) does reduction and progressively discarding attributes based on high or low dissonance values. Through this partitioning, similar objects will be compressed into separate tables that can be considered as representing coherent categories or meaningful descriptions. The final table will contain a generalized class description. It will be a much smaller table, covering all examples belonging to a certain class. This approach was intended to mine the database to establish broad regularities describing the database.

BDV is performed non-incrementally on an entire database represented as a flat file rather than taking actions on each record separately. It is a bottom-up discovery approach without domain knowledge. At the iteration, the dissonance will be computed for each attribute and compared with other values occurring within the current partition. They will not be compared with dissonance values

from other partitions. The dissonance value is calculated for each attribute by counting the number of unique attribute-values within the attribute.

Partitioning is done on attributes containing the least amount of dissonance greater than one. This approach develops a tree-like structure where each node has one parent and multiple child nodes. Traversals of the data structure are restricted to following parent to child nodes. No backtracking is allowed in this system. By enforcing these restrictions, the combinatorial explosion among the attributes is limited.

4.2.2 Dependencies and Aggregation

Occasionally, partitioning may not serve as an adequate method for reducing the given table. In these cases, another technique must be carried out to reduce the table. Aggregation, abstraction [Dhar, Tuzhilin 1993], and generalization techniques [Han 1992] in addition to partitioning are often used to reduce the search space. In the past, these techniques have been based on a priori domain knowledge. This knowledge has been used to reduce the number of attribute-values within a given attribute, allowing for an increase in the amount of redundant records within the table.

Past research has developed many algorithms for finding dependency networks within the database. Dependency information allows one attribute to be related to another. KDW [Piatetsky-Shapiro, Matheus 1992] discovers dependencies that measure the interdependencies of attributes. Not only can this information allow certain attributes to be predicted; it can also be used to form generalizations and abstractions within the database.

Dependencies play a large role in normalized databases. Current methods of data mining however, de-normalize these data sources and send a flat file into their algorithms. In guided data mining applications, dependencies can be encoded in domain knowledge [Han 1992], etc. In a pure unguided data mining algorithm, this information is ignored. Determining these current dependencies was not a primary focus of this study. In current algorithms, the issue of existing dependencies is avoided by creating flat files where known dependencies are not included. Therefore, a limitation of this algorithm is its inability to recognize existing dependencies within the data. When viewing the results of this algorithm, relations may seem to have been ignored or avoided because of the lack of prior knowledge use. However, unguided mining still offers advantages when different results are developed that may have been previously unknown.

If the database to be mined is a closed world where some attributes involved could be related, interdependencies among the attributes could be found or developed. BDV uses attributes containing a high coherence in the aggregation process based on the idea of interdependencies existing within the database. Whenever an attribute is removed because of a high level of coherence (i.e., containing only one attribute-value), its value is stored to be applied in future aggregations. Because BDV is an unguided approach containing no direct way of determining dependencies, every attribute value containing a high coherence level could possibly be used for abstraction of other attribute-values. These collected elements to be used for aggregations have been referred to in the BDV algorithm as abstraction sets.

4.2.3 Termination Requirements

Finally, for any system, the termination requirement must ensure that the algorithm stops. However, within data mining, this stopping point can directly affect the results of the system. It affects the size of the final table on which the complexity of the final if-then rule is dependent. Within data mining, most termination requirements are based on thresholds that mark the minimum amount of records to be maintained within the final table [Han 1992]. In addition, the user usually sets these threshold values before the data mining process begins [Piatetsky-Shapiro, Matheus 1991].

4.3 BDV Algorithm

The following steps are a detailed explanation of the BDV algorithm containing the main tasks discussed above.

1. Place the database to be mined into a flat file
2. Reduce dissonance within the table by:
    a. Calculate total number of examples within partition: T
    b. Calculate dissonance values within each attribute. $D(A_i)$ = Number of different values each attribute, $0<=i<=n$, where n is the number of attributes.
    c. If $D(A_i) = 1$ then eliminate attribute (attribute contains only one value - attribute has reached the maximum level of information value given to partition of table). Capture all potential abstraction information generated from A_i and save A_i and it's value.
    d. If $D(A_i) = T$ (are distinct values) and A_i is either of enumerated or ordered-enumerated type then
        - Eliminate attribute based on unique values containing little information.

e. If $Max[D(A_j)] < T$ and A_j is either enumerated or ordered-enumerated

- Check for redundancy in table - if records differ in one attribute $D(A_i)$ then perform the aggregation task.
- Eliminate the attribute in any abstraction set previously collected.

f. Check for redundancy within table - if two or more records contain the same values, reduce into a single line.

g. Find smallest $D(A_i) > 1$ and place the values of A_i into a set C.

h. If there are contributing attributes to C, select one attribute by applying a partitioning heuristic. If C is made up of a single contributing attribute, then use that attribute for partitioning.

i. Use the attribute found in step (h) to partition the table, each partition receiving a single value within A_i.

j. Repeat step 2 until the number of examples within the table meet the given termination requirement. If the resulting table reaches a point where no further partitioning or elimination can be done and the termination requirement has not been met, perform aggregation based on captured hierarchy sets. If there is a hierarchy set that can be applied and result in reducing the number of records within the partition, aggregation will be performed

3. If termination requirement has been met, end algorithm.

4. On the basis of results of algorithm, develop a representation of the knowledge learned.

4.4 Detailed Example: Record Album Data Set

An example that describes how the algorithm performs and what roles these three subtasks (partitioning, aggregation, and termination requirements) play in the overall algorithm follows. In addition, this example will demonstrate where potential experimentation/studies can be performed. The termination requirements set within this example create a minimum table containing at least one attribute with three records. It will be checked every time a new table is formed.

In addition, the number of records a particular instance covers is kept track of under the cover column. The cover column is a product of the algorithm. It is not part of the original flat file.

Id	*Group*	*Album Title*	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Sales*	*Covers*
1	REM	Monster	Rock/Pop	Warner Bros.	Nov1994	USA	yes	3,500,000	1
2	U2	Zooropa	Rock/Pop	Island	Apr1993	Foreign	no	3,432,000	1
3	REM	Out of Time	Rock/Pop	Warner Bros.	Jun1990	USA	yes	2,500,100	1
4	Nirvana	Nevermind	Rock/Pop	Geffen	Sept1989	USA	no	1,500,000	1
5	Nirvana	Unplugged	Rock/Pop	Geffen	Nov1994	USA	no	2,500,600	1
6	The Stone Roses	Second Coming	Alternative	Silvertone	Apr1995	Foreign	no	3,000,000	1
7	Pearl Jam	Ten	Rock/Pop	Epic	1991	USA	yes	1,450,300	1
8	Sting	Soul Cages	Rock/Pop	A&M	Apr1993	Foreign	no	2,250,000	1
9	The Cranberries	No Need to Argue	Alternative	Island	Apr1995	Foreign	yes	3,500,000	1
10	Pearl Jam	Vs.	Rock/Pop	Epic	Nov1992	USA	yes	1,500,300	1
11	James	Laid	Alternative	Mercury	Sept1993	Foreign	no	1,450,000	1
12	Squeeze	Frank	Alternative	A&M	1987	Foreign	no	1,200,000	1

Table 4.4.1: Original Music Album Table

Table 4.4.1 is the original table used in this example of the algorithm. It contains records for twelve music albums including attributes for artist, record label, release date, tour status, and sales.

The first step is to calculate the dissonance values within each attribute.

D = [12,9,12,2,7,9,11,2,2, 12] T = 12

Find the attribute(s) with the highest dissonance value. From these numbers, we can remove attribute 1 *Id* because it is the most dissonant collection. The attribute-value data type is enumerated and it is leftmost: D *(A1)=12=T.*

Group	*Album Title*	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Sales*	*Covers*
REM	Monster	Rock/Pop	Warner Bros.	Nov1994	USA	yes	3,500,000	1
U2	Zooropa	Rock/Pop	Island	Apr1993	Foreign	no	3,432,000	1
REM	Out of Time	Rock/Pop	Warner Bros.	Jun1990	USA	yes	2,500,100	1
Nirvana	Nevermind	Rock/Pop	Geffen	Sept1989	USA	no	1,500,000	1
Nirvana	Unplugged	Rock/Pop	Geffen	Nov1994	USA	no	2,500,600	1
The Stone Roses	Second Coming	Alternative	Silvertone	Apr1995	Foreign	no	3,000,000	1
Pearl Jam	Ten	Rock/Pop	Epic	1991	USA	yes	1,450,300	1
Sting	Soul Cages	Rock/Pop	A&M	Apr1993	Foreign	no	2,250,000	1
The Cranberries	No Need to Argue	Alternative	Island	Apr1995	Foreign	yes	3,500,000	1
Pearl Jam	Vs.	Rock/Pop	Epic	Nov1992	USA	yes	1,500,300	1
James	Laid	Alternative	Mercury	Sept1993	Foreign	no	1,450,000	1
Squeeze	Frank	Alternative	A&M	1987	Foreign	no	1,200,000	1

Table 4.4.2: Table after removal of unique attribute – ID

Once a new table is formed (i.e., a partition is made, an attribute is removed based on uniqueness or single attribute-value), the termination requirement is checked and the dissonance values are recalculated.

D=[9,12,2,7,9,11,2,2,12] T = 12

From this set, the attribute *Album Title* is leftmost and contains a dissonance level equal to the total number of records within the partition. In addition, the attribute-value is an enumerated data type, and consequently can be removed, resulting in Table 4.4.3.

Group	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Sales*	*Covers*
REM	Rock/Pop	Warner Bros.	Nov1994	USA	yes	3,500,000	1
U2	Rock/Pop	Island	Apr1993	Foreign	no	3,432,000	1
REM	Rock/Pop	Warner Bros.	Jun1990	USA	yes	2,500,100	1
Nirvana	Rock/Pop	Geffen	Sept1989	USA	no	1,500,000	1
Nirvana	Rock/Pop	Geffen	Nov1994	USA	no	2,500,600	1
The Stone Roses	Alternative	Silvertone	Apr1995	Foreign	no	3,000,000	1
Pearl Jam	Rock/Pop	Epic	1991	USA	yes	1,450,300	1
Sting	Rock/Pop	A&M	Apr1993	Foreign	no	2,250,000	1
The Cranberries	Alternative	Island	Apr1995	Foreign	yes	3,500,000	1
Pearl Jam	Rock/Pop	Epic	Nov1992	USA	yes	1,500,300	1
James	Alternative	Mercury	Sept1993	Foreign	no	1,450,000	1
Squeeze	Alternative	A&M	1987	Foreign	no	1,200,000	1

Table 4.4.3: Table after removal of unique attribute-values in Album Title

Again, with the formation of a new table, the termination requirement is checked and dissonance values are recalculated.

D=[9,2,7,9,11,2,2, 12] $T = 12$

The attribute with the highest dissonance value equal to the total number of records is *Sales*. It will be the next attribute to be removed, resulting in the partition shown in Table 4.4.4.

Group	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Covers*
REM	Rock/Pop	Warner Bros.	Nov1994	USA	yes	1
U2	Rock/Pop	Island	Apr1993	Foreign	no	1
REM	Rock/Pop	Warner Bros.	Jun1990	USA	yes	1
Nirvana	Rock/Pop	Geffen	Sept1989	USA	no	1
Nirvana	Rock/Pop	Geffen	Nov1994	USA	no	1
The Stone Roses	Alternative	Silvertone	Apr1995	Foreign	no	1
Pearl Jam	Rock/Pop	Epic	1991	USA	yes	1
Sting	Rock/Pop	A&M	Apr1993	Foreign	no	1
The Cranberries	Alternative	Island	Apr1995	Foreign	yes	1
Pearl Jam	Rock/Pop	Epic	Nov1992	USA	yes	1
James	Alternative	Mercury	Sept1993	Foreign	yes	1
Squeeze	Alternative	A&M	1987	Foreign	no	1

Table 4.4.4: Table after removal of unique attribute-values in Sales

The dissonance values are recalculated:

D=[9,2,7,9,11,2,2,12] $T = 12$

At this point, there are no attributes that can be due to unique values. In addition, there are no attributes that can be removed based on containing a single value. Partitioning must be performed on the attributes containing the least amount of dissonance (maximum coherence).

When there is more than one attribute containing the same minimum amount of dissonance, an additional partitioning heuristic must be evoked. Partitioning can be done in one of three ways: (a) based on balancing the resulting partitions (Table 4.4.5a), (b) unbalanced resulting partitions (Table 4.4.5b), or (c) by an arbitrary choice (Table 4.4.5c). In the example there are three attributes that meet this requirement: Type, *Citizen*, and *On Tour*. Because we have three heuristics to choose from, the algorithm will be tested with each heuristic (balanced, unbalanced and arbitrary, see Chapter 3 for discussion).

Group	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Covers*
REM	Rock/Pop	Warner Bros.	Nov1994	USA	yes	1
REM	Rock/Pop	Warner Bros.	Jun1990	USA	yes	1
Pearl Jam	Rock/Pop	Epic	1991	USA	yes	1
The Cranberries	Alternative	Island	Apr1995	Foreign	yes	1
James	Alternative	Mercury	Sept1993	Foreign	yes	1
Pearl Jam	Rock/Pop	Epic	Nov1992	USA	yes	1

Table 4.4.5a: Table after balanced heuristic used for partition

Group	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Covers*
REM	Rock/Pop	Warner Bros.	Nov1994	USA	yes	1
U2	Rock/Pop	Island	Apr1993	Foreign	no	1
REM	Rock/Pop	Warner Bros.	Jun1990	USA	yes	1
Nirvana	Rock/Pop	Geffen	Sept1989	USA	no	1
Nirvana	Rock/Pop	Geffen	Nov1994	USA	no	1
Pearl Jam	Rock/Pop	Epic	1991	USA	yes	1
Sting	Rock/Pop	A&M	Apr1993	Foreign	no	1
Pearl Jam	Rock/Pop	Epic	Nov1992	USA	yes	1

Table 4.4.5b: Table after unbalanced heuristic used for partition

Group	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Covers*
REM	Rock/Pop	Warner Bros.	Nov1994	USA	yes	1
REM	Rock/Pop	Warner Bros.	Jun1990	USA	yes	1
Pearl Jam	Rock/Pop	Epic	1991	USA	yes	1
The Cranberries	Alternative	Island	Apr1995	Foreign	yes	1
James	Alternative	Mercury	Sept1993	Foreign	yes	1
Pearl Jam	Rock/Pop	Epic	Nov1992	USA	yes	1

Table 4.4.5c: Table after arbitrary heuristic used for partition

To complete this example, we will follow the partitions created by the balanced and the unbalanced heuristics. When these new partitions are formed, we again check the termination requirements and calculate dissonance values:

Balanced Heuristic: D =[4,2,4,6,2,1] T = 6
Unbalanced Heuristic: D =[5,1,5,7,2,2] T = 8

Within the balanced partition, the smallest dissonance value calculated is one, occurring in the *On Tour* attribute. This attribute can now be removed because it does not contribute any more information to the current table (Table 4.4.6a). When attributes are removed based on a dissonance value of one, information for abstraction sets is collected (Listing 4.4.6a). In addition, the attribute name

and value removed is stored for use in the results of the algorithm. In this case, *On Tour* {yes} is stored as collected information.

Group	*Type*	*Record Label*	*Release Date*	*Citizen*	*Covers*
REM	Rock/Pop	Warner Bros.	Nov1994	USA	1
REM	Rock/Pop	Warner Bros.	Jun1990	USA	1
Pearl Jam	Rock/Pop	Epic	1991	USA	1
James	Alternative	Mercury	Sept1993	Foreign	1
The Cranberries	Alternative	Island	Apr1995	Foreign	1
Pearl Jam	Rock/Pop	Epic	Nov1992	USA	1

Table 4.4.6a: Table after removing attribute with single value, On Tour - Balanced Partition

On Tour {yes} ∈ Group {REM, Pearl Jam, James, The Cranberries}
On Tour {yes} ∈ Type {Rock/Pop, Alternative}
On Tour {yes} ∈ Record Label {Warner Bros., Epic, Mercury, Island}
On Tour {yes} ∈ Release Date {Nov1994, Jun1990, 1991, Sept1993, Apr1995, Nov1992}
On Tour {yes} ∈ Citizen {USA, Foreign}

Listing 4.4.6a: Collected Abstraction Sets from Table 4.4.6a

Within the unbalanced partition, the *Type* attribute contains a single value and can be removed (See Table 4.4.6b). Abstraction sets can be collected (Listing 4.4.6b) and *Type* {Rock/Pop} is stored as collected information.

Group	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Covers*
REM	Warner Bros.	Nov1994	USA	yes	1
U2	Island	Apr1993	Foreign	no	1
REM	Warner Bros.	Jun1990	USA	yes	1
Nirvana	Geffen	Sept1989	USA	no	1
Nirvana	Geffen	Nov1994	USA	no	1
Pearl Jam	Epic	1991	USA	yes	1
Sting	A&M	Apr1993	Foreign	no	1
Pearl Jam	Epic	Nov1992	USA	yes	1

Table 4.4.6b: Table after removing attribute with single value, Type Unbalanced Partition

Type {Rock/Pop} ∈ Group{REM, U2, Nirvana, Pearl Jam, Sting}
Type {Rock/Pop} ∈ Record Label {Warner Bros., Island, Geffen, Epic, A&M}
Type {Rock/Pop} ∈ Release Date {Nov1994, Apr1993, Jun1990, Sept1989, 1991, Apr1993, Nov1992}
Type {Rock/Pop} ∈ Citizen {USA, Foreign}
Type {Rock/Pop} ∈ On Tour {yes, no}

Listing 4.4.6b: Collected Abstraction Sets from Table 4.4.6b

When new tables have been formed, check termination requirements and calculate dissonance values:

Balanced Heuristic: D=[4,2,4,6,2] T = 6
Unbalanced Heuristic: D=[5,5,7,2,2] T = 8

Within the balanced partition, the highest dissonance value, found in the attribute *Release Date*, is equal to the total number of records within the partition. The values within this attribute can be considered unique and can be removed from the table. Results of this removal are shown in Table 4.4.7a.

Group	*Type*	*Record Label*	*Citizen*	*Covers*
REM	Rock/Pop	Warner Bros.	USA	1
REM	Rock/Pop	Warner Bros.	USA	1
Pearl Jam	Rock/Pop	Epic	USA	1
James	Alternative	Mercury	Foreign	1
The Cranberries	Alternative	Island	Foreign	1
Pearl Jam	Rock/Pop	Epic	USA	1

Table 4.4.7a: Table after removing attribute with unique value, Release Date - Balanced Partition

Within the unbalanced partition, there are no attributes with dissonance value equal to the total amount of records or a dissonance value equal to one. Therefore, this table must be partitioned. Again, there is more than one attribute having the least amount of dissonance: Type and *Citizen*. *Citizen* is chosen because it will result in the most unbalanced partitions. One unbalanced partition is shown in Table 4.4.7b.

Group	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Covers*
REM	Warner Bros.	Nov1994	USA	yes	1
REM	Warner Bros.	Jun1990	USA	yes	1
Nirvana	Geffen	Sept1989	USA	no	1
Nirvana	Geffen	Nov1994	USA	no	1
Pearl Jam	Epic	1991	USA	yes	1
Pearl Jam	Epic	Nov1992	USA	yes	1

Table 4.4.7b: Table after Partitioning on Citizen Attribute - Unbalanced Partition

A new table has been created. Check termination requirements and calculate dissonance values.

Balanced Heuristic: D=[4,2,4,2] T = 6
Unbalanced Heuristic: D=[3,3,5,1,2] T = 6

Within the balanced partition, redundancies are eliminated before any other checks are performed (Table 4.4.8a).

Group	*Type*	*Record Label*	*Citizen*	*Covers*
REM	Rock/Pop	Warner Bros.	USA	2
The Cranberries	Alternative	Island	Foreign	1
James	Alternative	Mercury	Foreign	1
Pearl Jam	Rock/Pop	Epic	USA	2

Table 4.4.8a: Table after eliminating redundancies - Balanced Partition

Within the unbalanced partition, the attribute *Citizen* contains a single value, dissonance equal to one. Therefore, it is removed based on single values resulting in Table 4.4.8b. Abstraction sets are collected (Listing 4.4.8b) and appended and *Citizen* {USA} is stored as collected information.

Group	*Record Label*	*Release Date*	*On Tour*	*Covers*
REM	Warner Bros.	Nov1994	yes	1
REM	Warner Bros.	Jun1990	yes	1
Nirvana	Geffen	Sept1989	no	1
Nirvana	Geffen	Nov1994	no	1
Pearl Jam	Epic	1991	yes	1
Pearl Jam	Epic	Nov1992	yes	1

Table 4.4.8b: Table after removing a single value - Unbalanced Partition

Citizen {USA} ∈ Group {REM, Nirvana, Pearl Jam}
Citizen {USA} ∈ Record Label {Warner Bros., Geffen, Epic}
Citizen {USA} ∈ Release Date {Nov1994, Jun1990, Sept1989, 1991, Nov1992}
Citizen {USA} ∈ On Tour {yes, no}

Listing 4.4.8b: Abstraction sets collected from Table 4.4.8b

Check termination requirement. If requirement is not met, calculate dissonance values.

Balanced Heuristic: D =[4,2,4,2] T = 4
Unbalanced Heuristic: D =[3,3,5,2] T = 6

Within the balanced partition, an attribute with unique values exists, *Group*. Remove the *Group* attribute-values because of these unique values (Table 4.4.9a).

Type	*Record Label*	*Citizen*	*Covers*
Rock/Pop	Warner Bros.	USA	2
Alternative	Island	Foreign	1
Alternative	Mercury	Foreign	1
Rock/Pop	Epic	USA	2

Table 4.4.9a: Table after eliminating unique attribute values - Balanced Partition

In the unbalanced partition, there are no attributes where the dissonance value is equal to the total number of records. In addition, there are no attributes where

the minimum dissonance value is equal to two. Therefore, partitioning can be performed *On Tour*. (Table 4.4.9b)

Group	*Record Label*	*Release Date*	*On Tour*	*Covers*
REM	Warner Bros.	Nov1994	yes	1
REM	Warner Bros.	Jun1990	yes	1
Pearl Jam	Epic	1991	yes	1
Pearl Jam	Epic	Nov1992	yes	1

Table 4.4.9b: Table after partitioning On Tour- Unbalanced Partition

New tables have been formed. Check termination requirement and calculate dissonance values.

Balanced Heuristic: D=[2,4,2] T= 4
Unbalanced Heuristic: D=[2,2,4,2] T = 4

Within the balanced partition, the *record label* attribute can be removed because its dissonance value is equal to the total number of records (Table 4.4.10a).

Type	*Citizen*	*Covers*
Rock/Pop	USA	2
Alternative	Foreign	1
Alternative	Foreign	1
Rock/Pop	USA	2

Table 4.4.10a: Table after eliminating unique attribute values - Balanced Partition

In the unbalanced partition, *On Tour* contains a single value. This attribute can be removed (Table 4.4.10b), abstraction sets can be collected (Listing 4.4.10b), and the attribute-value can be collected as one of the results.

Group	*Record Label*	*Release Date*	*Covers*
REM	Warner Bros.	Nov1994	1
REM	Warner Bros.	Jun1990	1
Pearl Jam	Epic	1991	1
Pearl Jam	Epic	Nov1992	1

Table 4.4.10b: Table after removing On Tour- Unbalanced Partition

On Tour {yes} ∈ Group {REM, Pearl Jam}
On Tour {yes} ∈ Record Label {Warner Bros., Epic}
On Tour {yes} ∈ Release Date {Nov1994, Jun1990, 1991, Nov1992}

Listing 4.4.10b: Abstraction Sets Collected from Table 4.4.10b - Unbalanced Partition

New tables have been created. Check termination requirements and calculate dissonance values.

Balanced: D=[2,2] T=4
Unbalanced: D=[2,2,4] T = 4

In the balanced table, before performing partitioning, the table is checked for redundancies. Since there are redundant records within this table, they are collapsed to form Table 4.4.11a.

Type	*Citizen*	*Covers*
Rock/Pop	USA	4
Alternative	Foreign	2

Table 4.4.11a: Table after eliminating redundant data - Balanced Partition

Within the unbalanced table, there is an attribute with a dissonance value equal to the total number of records within the partition *Release Date*. Table 4.4.11b will result when this attribute is removed.

Group	*Record Label*	*Covers*
REM	Warner Bros.	1
REM	Warner Bros.	1
Pearl Jam	Epic	1
Pearl Jam	Epic	1

Table 4.4.11b: Table after removing Release Date - Unbalanced Partition

New tables have been created. Check for termination requirement. At this point, the balanced table meets the termination requirement: consisting of less than three records and maintaining at least one attribute field. However, the unbalanced table still has more than three records. Therefore, only the dissonance values are computed for the unbalanced table.

Unbalanced: D=[2,2] T = 4

As in the previous step for the balanced table, before partitioning is performed on the attributes containing a dissonance value of two, a check for redundant records is made. Redundant records are found and then collapsed forming Table 4.4.12.

Group	Record Label	Covers
REM	Warner Bros.	2
Pearl Jam	Epic	2

Table 4.4.12: Table after removing Redundant Records - Unbalanced Partition

At this point, the table meets the termination requirement. Table 4.4.11a is the final table for the balanced heuristic while Table 4.4.11b is the final table for the unbalanced heuristic.

The products of these algorithms include collected information; a final table, the complete abstraction sets, and an if-then rule based on the products.

Balanced Heuristic	Unbalanced Heuristic
On Tour {yes}	On Tour {yes}
	Citizen {USA}
	Type {Rock/Pop}

Table 4.4.13: Collected Information

Balanced Heuristic			Unbalanced Heuristic	
Type	Citizen		Group	Record Label
Rock/Pop	USA		REM	Warner Bros.
Alternative	Foreign		Pearl Jam	Epic

Table 4.4.14: Final Tables for both Balanced and Unbalanced Heuristic

Balanced Heuristic

On Tour {yes} ∈ Group {REM, Pearl Jam, James, The Cranberries}
On Tour {yes} ∈ Type {Rock/Pop, Alternative}
On Tour {yes} ∈ Record Label {Warner Bros., Epic, Mercury, Island}
On Tour {yes} ∈ Release Date {Nov1994, Jun1990, 1991, Sept1993, Apr1995, Nov1992}
On Tour {yes} ∈ Citizen {USA, Foreign}

Unbalanced Heuristic

Type {Rock/Pop} ∈ Group {REM, U2, Nirvana, Pearl Jam, Sting}
Type {Rock/Pop} ∈ Record Label {Warner Bros., Island, Geffen, Epic, A&M}
Type {Rock/Pop} ∈ Release Date {Nov1994, Apr1993, Jun1990, Sept1989, 1991, Apr1993, Nov1992}
Type {Rock/Pop} ∈ Citizen {USA, Foreign}
Type {Rock/Pop} ∈ On Tour {yes, no}
Citizen {USA} ∈ Group {REM, Nirvana, Pearl Jam}
Citizen {USA} ∈ Record Label {Warner Bros., Geffen, Epic}
Citizen {USA} ∈ Release Date {Nov1994, Jun1990, Sept1989, 1991, Nov1992}
Citizen {USA} ∈ On Tour {yes, no}
On Tour {yes} ∈ Group {REM, Pearl Jam}
On Tour {yes} ∈ Record Label {Warner Bros., Epic}
On Tour {yes} ∈ Release Date {Nov1994, Jun1990, 1991, Nov1992}

Listing 4.4.14: Collected Abstraction Sets

Balanced Heuristic	Unbalanced Heuristic
IF On Tour {yes} THEN Type {Rock/Pop} and Citizen {USA} OR Type {Alternative} and Citizen {Foreign}	IF Citizen {USA} and Type {Rock/Pop} and On Tour {yes} THEN Group {REM} and Record Label {Warner Bros.}

Table 4.4.15: Final relationship rule for balanced and unbalanced heuristic

From this example, there is a difference between implementations using different partitioning techniques. While the aggregation techniques were just

barely touched on, it seems safe to assume they will become important as the data sets become larger.

From the products of this algorithm, it has not been determined which partitioning heuristic is better or worse. In fact, this example can demonstrate the difficulty in determining what the better algorithm is. This type of decision or conclusion cannot be reached based only on products of the algorithm. Rather, information must be collected throughout execution to support one claim or another.

4.5 Blind Dissonant Value Data Mining [BDV]

This algorithm mines the database by reducing the data table through iterations. The reduction is based on partitioning the table using the attribute with the lowest dissonance value. When more than one attribute contains the minimum dissonant value, a heuristic for determining the attribute to partition on has been developed and tested. Partitioning will be done if (a) the tables have not reached a termination requirement or (b) the tables contain an attribute meeting the requirements for partitioning.

The algorithm's aggregation task reduces the number of records within the table by combining similar records. The abstraction task uses information collected throughout the execution of the BDV algorithm. When abstraction is applied, the resulting value contains all possible values that could have been applied because the system is not aware of the best value to abstract. This allows the

user to decide at the conclusion of the algorithm which value makes the most sense for abstraction purposes.

Termination requirements are used to signal the completion of the algorithm. These requirements are checked every time a new table/partition is formed within the algorithm. Termination requirements have been found to vary according to the size of the data set. Therefore, the termination requirement(s) has been tested and developed using different data sets.

The BDV algorithm is given the types of data contained in the attributes of the data sets. Three data types exist: enumerated, enumerated-ordered, and non-scalar. This type of knowledge is needed to find techniques that can be done throughout the mining process. The knowledge will come from the data dictionary defining the original table. The data dictionary will contain the name of the attribute, the field width of the attribute (I/O purposes), and the data type. This will meet the constraints of our unguided strategy because the information is used to maintain execution of the learning process rather than limiting the search space.

The BDV algorithm will be successful if some meaningful pattern representing a relationship among data items can be discovered from the database. These relationships may not be new or interesting because there is no way for the BDV algorithm to determine newness or interestingness. Additionally, this study will be considered useful if it leads to new outlooks and possibilities leading to the improvement of the original BDV algorithm. Finally, the results of the system

will be considered correct if they produce valid relations. That is, the results will be considered correct if they describe records within the database.

The focus of this project was to find methods that can best accomplish blind dissonance mining. The investigation was organized by dividing the algorithm into subtasks: partitioning, aggregation, and termination. These subtasks were developed and compared through testing on various data sets.

4.5.1 Measuring Dissonance

The BDV algorithm is based on the manipulation of dissonance values. For this study, dissonance values represent the disorder within an attribute. These values are calculated by counting the number of unique attribute-values within an attribute (1). The limits of the dissonance values are stated in (2).

(1) $D(A_i)$ = number of unique attribute-values in A_i; where A_i is an attribute column, I being $0<i<n$ where n= total number of attributes.

(2) $0< D(A_i) < T$ where T = total number of records within a table.

The study focused on this measure of dissonance because of its simplicity. Because this was a first study into the BDV algorithm, there were many areas that could be studied. The measurement, while playing a major role in the algorithm, also plays a role that could be easily replaced by a more specific and complex measurement. Other tasks within the algorithm would be more difficult to replace, such as the aggregation techniques or partitioning implementations. In future versions of the algorithm, to provide a more

thorough testing, the dissonance measure could be replaced with well-studied methods.

This number can also represent the level of coherence within an attribute. An attribute containing many identical attribute-values will have a high level of coherence represented by a low dissonance value. In the same way, those attributes containing many unique attribute-values will have a high dissonance value (representing a high level of disorder). Attributes with a low dissonance value are assumed to contain information. On the other hand, attributes with high dissonance values are assumed to contain limited information. Dissonance values are compared within the current table/partition. From this comparison, actions within the algorithm are taken.

4.5.2 Partitioning Heuristic

In the BDV algorithm, forming new tables that are subsets of the previous tables reduces the search space. These new tables are called partitions because each new table represents a given part of the original table. In addition, these partitions are used to increase the informative value of the search space. Partitions are formed on the attribute containing the minimum dissonance value.

The tie-breaking partitioning heuristic is used to choose an attribute to partition on when more than one attribute contains the same minimum dissonant value. The tie-breaker is based on the ability to look ahead to find the size of the future partitions. If the size of the future partition is close to half the size of the

current partition, then it has been hypothesized that the partition will contain more information that is coherent.

For example, if there is a tie between two attributes containing a minimum dissonance value, the partitioning heuristic will use information about the number of records in the resulting partition to break the tie. If the resulting partition contains approximately *T/2*, where *T* = total records of current table/partition, then it is a balanced partition. If the resulting table/partition contains much less than *T/2* or greater than *T/2,* then the resulting partition is unbalanced.

The experimental question explored was to verify that maintaining balanced partitions throughout the execution of the algorithm would increase the information within the resulting table. For verification, three variations are investigated.

1. *Balanced*: Select the attribute that would result in the most balanced partitions. In Table 4.4.1, the first partition will be performed on *On Tour* because $D(A_j) = 2$. The future partition contains *Total/2* records as shown in Table 4.5.2.1.

Id	*Group*	*Album Title*	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Sales*
1	REM	Monster	Rock/Pop	Warner Bros.	Nov1994	USA	yes	3,500,000
3	REM	Out of Time	Rock/Pop	Warner Bros.	Jun1990	USA	yes	2,500,100
7	Pearl Jam	Ten	Rock/Pop	Epic	1991	USA	yes	1,450,300
9	The Cranberries	No Need to Argue	Alternative	Island	Apr1995	Foreign	yes	3,500,000
11	James	Laid	Alternative	Mercury	Sept1993	Foreign	yes	1,450,000
10	Pearl Jam	Vs.	Rock/Pop	Epic	Nov1992	USA	yes	1,500,300

Table 4.5.2.1: Music Album Data Set - Balanced Heuristic

2. *Unbalanced*: Select the attribute that would result in the most unbalanced partition. In Table 4.4.1, the first partition will be performed on *Type* because $D(A_j) = 2$. The resulting partition is enumerated in Table 4.5.2.2.

Id	*Group*	*Album Title*	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*	*Sales*
1	REM	Monster	Rock/Pop	Warner Bros.	Nov1994	USA	yes	3,500,000
2	U2	Zooropa	Rock/Pop	Island	Apr1993	Foreign	no	3,432,000
3	REM	Out of Time	Rock/Pop	Warner Bros.	Jun1990	USA	yes	2,500,100
4	Nirvana	Nevermind	Rock/Pop	Geffen	Sept1989	USA	no	1,500,000
5	Nirvana	Unplugged	Rock/Pop	Geffen	Nov1994	USA	no	2,500,600
7	Pearl Jam	Ten	Rock/Pop	Epic	1991	USA	yes	1,450,300
8	Sting	Soul Cages	Rock/Pop	A&M	Apr1993	Foreign	no	2,250,000
10	Pearl Jam	Vs.	Rock/Pop	Epic	Nov1992	USA	yes	1,500,300

Table 4.5.2.2: Music Album Data Set - Unbalanced Heuristic

3. *Arbitrary choice between candidates*: Select an attribute regardless of the resulting partitions. In Table 4.4.1, the first partition will be performed on one of three attributes containing the minimum dissonant value of 2: Type, *Citizen,* or *On Tour*.

4.5.3 Aggregation Technique

When eliminating an attribute based on a dissonance value of one, the attribute and value can be used to define abstraction sets. These sets may be applied at certain points within the algorithm for aggregation purposes. These abstraction sets will only be generated for enumerated data types. Scalar data types can possibly be dealt with using approximate methods such as fuzzy, rough sets, or Dempster-Shafer methods. To allow attributes with common values among themselves, the abstraction sets save not only the attribute values, but also the name of their related attribute. By storing this information at this step, it can be used for the final formulation of the rule and potentially a better aggregation.

Aggregation will be performed when a value's dissonance is less than the total number of records but greater than two occurs. Aggregation is used to reorganize the data within the table. It was hypothesized that aggregation would produce redundant records by "aggregating" an attribute and decreasing its dissonance value to such an extent that duplicate records appear. These duplicate records will then be compressed resulting in a table that will meet or exceed the termination requirement.

Aggregation will use the abstraction sets created and stored throughout the BDV algorithm. Because determining which element of an abstraction set to use is difficult and not yet satisfactorily developed, a variety of methods can be used to overcome this difficulty.

Allow the user to view the potential hierarchies and let he or she decide which sets to apply.

Generate multiple tables by applying different elements or combinations of the abstraction sets (this could prove unfeasible because of size constraints).

Develop a heuristic to determine which sets are applicable by using characteristics of the current table/partition, which attribute aggregation would generate the most information, i.e., generate the highest number of duplicate records, or generate attributes that maintain a dissonance value of one.

It also could be based on some type of comparison among the abstraction sets themselves. As an initial reaction, this may be complicated because the search space is not well contained or restricted to any size.

For experimental purposes, this algorithm does a combination of the first and third methods. It allows the user to decide what aggregation sets are applicable (at the end of the process) and inserts the maximum combination of abstraction sets into the aggregation. As a result, any attribute that has been aggregated using the captured abstraction sets lists all possible values found in that set to allow the user to determine the correct meaning.

For example, see Table 4.5.3.1. It represents the current partition for this example. Up to this point, the data mining process has collected *Type* (Rock/Pop) and *Citizen* (USA) as abstraction sets because they contain high levels of coherence (See Listing 4.5.3.2). Aggregation performed on this table would be done on the *Record Label* attribute because it contains the highest level of dissonance. By aggregating this attribute, redundant records will appear within the table. The BDV algorithm cannot determine which abstraction set would be the best one to use for the aggregation. Therefore, it will apply a combination of the abstraction sets in order for the user to determine the most applicable value to use at the end. The resulting partition is shown in Table 4.5.3.3.

Group	*Record Label*	*On Tour*
REM	Warner Bros.	Yes
REM	IRS	Yes
Nirvana	Geffen	No
Nirvana	Geffen	No
Pearl Jam	Warner Bros.	Yes
Pearl Jam	Epic	Yes

Table 4.5.3.1: Music Album Data Set - before aggregation

Type {Rock/Pop} ∈ Group {REM, Nirvana, PearlJam}
Type {Rock/Pop} ∈ On Tour {yes, no}
Type {Rock/Pop} ∈ Record Label {Warner Bros., IRS, Geffen, Epic}

Citizen {USA} ∈ Group {REM, Nirvana, PearlJam}
Citizen {USA} ∈ On Tour {yes, no}
Citizen {USA} ∈ Record Label {Warner Bros., IRS, Geffen, Epic}

Listing 4.5.3.2: Listing of collected abstraction sets

Collect: Record Label {Rock/Pop || USA}

Group	*On Tour*
REM	yes
Nirvana	no
PearlJam	yes

Table 4.5.3.3 Music Album Data Set - after aggregation

4.5.4 Termination Requirement

A termination requirement is also experimentally considered. When the termination requirement is met or exceeded, the algorithm will end. Usually, the termination requirement is set as a number of minimum records that must exist within the final table. Each record within a table is a concept of the model world. A table can be viewed as a large set of such concepts. The number of records within the final table will correspond to the number of concepts combined to form a description of the database. The more records included in the final table, the more semantically complex. To limit the complexity of this description, the termination requirement should set a minimum number of records to be within the final table. This number should be small (usually two to eight). A large minimum may lead to a complex semantic rule where the results may not be fully generalized. On the other hand, a small minimum leads to a simple rule with few semantic concepts. However, small minimums may result in a generalized description and a loss of valuable information. DBLearn [Han 1992] solves the problems associated with termination requirements by allowing

the user to control the termination values of the algorithm. It was hypothesized that through this project, a suitable termination requirement could be developed. With the termination requirement fixed, the need for user interaction during the algorithm will be eliminated.

However, setting the termination requirement as a minimum number of records does not guarantee that the final table lack redundant attribute values. The importance and use of the contents of the final table must be determined. If the resulting knowledge is to be implemented as an if-then rule (as above), maintaining a number of attributes within the final table will be necessary. Therefore, the minimum number of attributes that must occur in the final table is also set. Finally, the termination requirement will include a flag to represent if the current table has been updated through partitioning, unique value removal, or single attribute-value removal. This flag (change flag) ensures that an infinite loop will be avoided if a table reaches a point where no further partitioning or aggregation can be done. For example, in the music album data set, the final table consists of three records (see Table 4.5.4.1). Through the execution of the algorithm, two values have been collected for the description: *Citizen* = USA and *Type* = Rock/Pop. The final description follows in diagram 4.5.4.2.

Group	*Record Label*	*On Tour*
REM	Warner Bros.	yes
Nirvana	Geffen	no
Pearl Jam	Epic	yes

Table 4.5.4.1: Final Table from Music Album Data set

IF
Citizen {USA} and *Type* {Rock/Pop}
THEN
Group {REM} and *RecordLabel* {Warner Bros.} and *On Tour* {yes}
Or
Group {Nirvana} and *RecordLabel* {Geffen} and *On tour* {no}
Or
Group {PearlJam} and *RecordLabel* {Epic} and *On Tour* {yes}

Diagram 4.5.4.2: Final Relationship converted to if-then format

4.5.5 Covers

To maintain an accurate description of the database, a value representing about the number of instances covered by a certain record is kept. Whenever redundant records are collapsed, a counter is incremented according to the number of records that were collapsed. This counter is called a "cover." In other words, if there are two identical records within the table, one of the matching records can be deleted and accounted for by incrementing the cover on the remaining record. The total number of covers within the final table will tell the user how many instances can be "covered" by the resulting knowledge.

Chapter 5

RESULTS AND ANALYSIS

5.1 Groundwork

The purpose of this study was to find the "best way" to implement the Blind Dissonance Value algorithm. To reach this goal, it was necessary to try different implementations and develop a way to compare their results. In addition to creating different implementations, different data sets were used for testing. This takes into account certain performance issues that arise when the test data set is large.

This study involved comparing multiple versions of the same algorithm with multiple data sets. For all versions of the algorithm the same metrics were collected. These form the basis of the comparisons done to find a conclusion.

5.1.1 Metrics

The metrics developed were chosen on the basis that they could be counted. This eliminated any subjective user qualifications from influencing the final analysis. Whether or not restricting the analysis of this algorithm to quantitative measurements is enough to support the conclusion is still questionable. For the purposes of this study, it was considered enough.

The following metrics was collected when the algorithm was tested. (For a further discussion on the metrics, see Appendix A).

Partitions: The number of times the algorithm splits the original table. I have assumed that every time the algorithm performs a partition, it is doing work.

Partitioning Trace: This is a diagram of the partitioning heuristic's performance.
Depth: Relates to the number of partitions needed to reach the final relation.
Breadth: Relates to the number of unresolved ties created by the partitioning heuristic.

Aggregations: The number of attributes that are summarized for a higher level of generalization.

Dependencies: The number of times a dependency occurs. A dependency occurs in this algorithm when there are two or more attributes containing a dissonance value of 1 (one attribute-value). Dependencies contain information.

Final Table Attributes: The number of attributes contained within the last table. This value relates to the semantic complexity of the final description's consequent

Number of Records Covered: The number of records from the original data set that are represented by the final relation.

Percentage Covered: The percentage of original records that are represented by the final relation.

Collected Information: The set of attribute values that have been collected to be used within the antecedent of the final relation.

Useful Information: A subset of collected information. It contains only the collected information sets that are not based on aggregated values. This information can be used as a minimal description.

5.1.2 Data Sets

The BDV algorithm was tested and developed using a variety of data sets. The data sets involved mostly nominal data types, from eight to twenty-five attributes, and from eight to thousands of records. More than one data set was used in an attempt to eliminate data dependent results to focus the study on the algorithm.

1. University Example: Student information containing eight records with seven attributes. It is based on previous work by [Han 1992]. It has been included within this study to form a comparison and validate the actions of the BDV algorithm.

2. Music Album Example: Contains eight records with seven attributes, all of nominal type. This example is used to describe the properties and actions taken by the algorithm on a small data set. It is a product of this study.

3. Pittsburgh Bridges Data Set: Containing eighty-two records and thirteen attributes. This data set originated in the department of Civil Engineering and Engineering Design Research Center at Carnegie Mellon University. It has been used in the development of incremental learning systems with mixed property types for engineering design purposes [Reich, 1989]. For this algorithm, all values were considered nominal.

4. Mushroom Database: The original mushroom database was made up of more than 8,000 records and 23 attributes all containing nominal values [Schlimmer, 1987]. For testing reasons, it has been broken into data sets of 500, 1,000 and 3,000 records (Mushroom_500, Mushroom_1000, and Mushroom_3000). These records have been drawn from the Audubon Society Field Guide to North American Mushrooms (1981). It has been used in many machine learning papers for incremental concept acquisition and learning. For example, by Schlimmer, [1987] and Iba, Wogulis, and Langley [1988].

5. EPA-RREL (Environmental Protection Agency, Risk Reduction Engineering Lab) Waste Water Treatability Database: For this study, the data set is based on a combination of EPATT and EPA Solid files. The data set included 18 attributes and was broken into two data sets, on containing 2,000 and the other 3,000 records. The RREL Treatability Database was designed to provide a thorough review of proven treatment technologies in the removal and/or destruction of chemicals in various media. This media includes municipal and industrial wastewater, drinking water, groundwater, soil, debris, sludge and sediments. Version 4.0 of the database was released in 1992. It contains 1,166 chemical compounds and more than 92,000 sets of waste water treatability data. [U.S. EPA Risk Reduction Engineering Laboratory, Cincinnati, 1995]

Except for the EPA-RREL data set, these data sets, especially the mushroom set, have been used in machine learning experiments and served as a well-known domain meeting the restrictions within the algorithm. (I.e., little noise and non-scalar attribute values). The EPA-RREL data set serves as the real world example used in the experimentation.

5.1.3 Partitioning Heuristics

The hypothesis on partitioning heuristics was that balanced partitions would contain more information than unbalanced. To prove or disprove this theory, three partitioning heuristics were developed, as listed below:

Balanced heuristic: produces multiple partitions where each partition to be created contains an equal percentage of the number of records from the current table.

Unbalanced heuristic: produces multiple partitions where each partition to be created contains the maximum unbalanced percentages of the number of records from the current table.

Arbitrary heuristic: produces partitions of the current table without regard to the percentages of records contained in the future partitions.

The balanced heuristic was included to prove the theory. The unbalanced heuristic was included to disprove the theory. Finally, the arbitrary heuristic was included to provide a basis for comparison.

5.1.4 Sequencing of Subtasks

When developing a new algorithm, it is necessary to form comparisons to similar areas to judge the validity, productivity, or efficiency of one to the other. Since my approach was rather different in a new field, and because this project's goal was to develop a new algorithm, comparisons were made against versions of the original BDV algorithm. These versions differed not only in the partitioning heuristics used, but also in the organization of the partitioning subtask and the aggregation subtask. These two subtasks form the major components of the algorithm. This idea lead to the formation of a grid that is pictured in diagram 5.1.4.1.

Early Aggregation, Limited Partitioning	Early Aggregation, Unlimited Partitioning
Late Aggregation, Limited Partitioning	Late Aggregation, Unlimited Partitioning

Diagram 5.1.4.1: Basic Grid outlining study

Within each cell of the grid, each partitioning heuristic was implemented for each data set. By using this grid, I was able to form some type of comparison that could be used to report and form conclusions.

5.1.5 Characteristics of a "Good" Result

When this project originated, the only requirement for a "good" result was a final relation that contained some type of informative value that was developed from a small number of partitions. This informative value in essence is the median value taken from overly specific and over generalized results. After achieving these objectives using the metrics developed, the following conclusions were made. A good result is characterized by:

Partitioning Traces: a low depth value, and a high breadth level. A shallow tree will be assumed to have completed the least amount of work to reach the conclusion. A wide tree, while pointing out failures within the tie-breaking scheme may also give more information in multiple relations.

A medium amount of useful and collected information. Data mining is judged according to the level of "interestingness" and informative content of its results. Through this study, it was found that small amounts of useful and collected information lead to very specific, narrow relations. When large numbers of useful and collected information occurred, the results contained over generalized relations.

A small number of partitions created by the algorithm. (The number of partitions created is directly related to the amount of work that the system performed.)

A medium number of aggregations performed within the algorithm. The number of aggregations performed is related to the amounts of useful and collected information developed throughout the algorithm.

A high number of dependencies found within the data set. It is elementary that dependencies, when found within databases contain a lot of information. Therefore, if a high number of dependencies is found the algorithm can be considered good.

A relation that covers a medium percentage of the original records using a medium number of attributes. Middle ground within the ranges of these two numbers is ideal due to the problem of relations being either too specific or too general. (See above)

5.2 Results and Conclusions

Using Diagram 5.1.4.1, we can develop two resulting tables: one for the subtask comparison (Table 5.2.1) and one for the comparison of the partitioning heuristics within the subtasks (Table 5.2.2). The cells that are gray within the grids met the criteria for a "good" result.

Early Aggregation - Limited Partitioning	**Early Aggregation - Limited Partitioning**
Partitioning Traces Depth – Low Breadth – Low	Partitioning Traces Depth – Low Breadth – Low
Aggregations - Tie	Aggregations - Tie
Dependencies - Tie	Dependencies - Tie
Useful Information - Low	Useful Information – Low
Collected Information - NA	Collected Information - NA
Late Aggregation - Limited Partitioning	**Late Aggregation – Unlimited Partitioning**
Partitioning Traces Depth – High Breadth – High	Partitioning Traces Depth – High Breadth – High
Aggregations - Tie (EPA only)	Aggregations – None
Dependencies - Balanced	Dependencies – Balanced
Useful Information – Medium	Useful Information – High
Collected Information – Medium	Collected Information - Low

Table 5.2.1 Comparisons based on Subtasks

Early Aggregation – Limited Partitioning	**Early Aggregation - Unlimited Partitioning**
Partitions - Balanced/Arbitrary	Partitions - Balanced/Arbitrary
Final Table Attributes – Balanced	Final Table Attributes - Balanced
% Records Covered – Balanced	% Records Covered - Balanced
Late Aggregation – Limited Partitioning	**Late Aggregation - Unlimited Partitioning**
Partitions – Balanced	Partitions - Balanced
Final Table Attributes - Balanced/Unbalanced	Final Table Attributes- Balanced/Unbalanced
% Records Covered – Tie	% Records Covered - Unbalanced

Table 5.2.2 Comparisons based on Partitioning Heuristics

5.2.1 Partitioning

The balanced heuristic performed very well based on the number of partitions created. The use of the other two heuristics was successful in this study because a difference appeared among them. These differences support the original

hypothesis that different results could be generated based solely on the partitioning heuristic implemented.

An issue that appeared throughout this study dealt with the value to use as a threshold to limit partitioning. Partitioning limits were based directly on the minimum dissonance value of an attribute. The original setting of this value was two. This setting assumes that there will always be an attribute that contains no more than two distinct values. As the mushroom data set proved, this is not always the case. In that data set, the minimum number of distinct values within an attribute was three. Here, the algorithm terminated as expected, but the result was neither interesting nor informative. Because of this slight dependency on the data set, the user should set the partitioning heuristic before the algorithm is started.

5.2.2 Sequencing of Subtasks

The best sequencing of subtasks within this study was shown to be late aggregation and limited partitioning. This sequence of tasks produced results that fell between the two extremes. The worst versions of the algorithm were those that contained early aggregation, regardless of what type of partitioning was used. The resulting sets generated using early aggregation were so overly generalized that very little useful information, even in the smaller data sets, was generated.

Throughout the study, as the result comparison tables show (Table 5.2.1 and Table 5.2.2), a difference appeared among the multiple versions of the algorithm. As the study progressed it became clear that there was no difference between the version containing early aggregation and limited partitioning and the version containing early aggregation and unlimited partitioning. This can be attributed to the cascading effect that the early aggregation had on the intermediate tables. The partitioning subtask was rarely, if ever reached within these versions.

The cascading effects found in the early aggregation versions of the algorithm bring up the issue of aggregation methods within the BDV algorithm. The BDV algorithm was not a study in aggregation methods; these were to be addressed in later research. The aggregation method used within this study was very primitive. To improve the results of this algorithm, a more sophisticated method of aggregation must be developed.

5.3 Complexity Study of Algorithm

For the complexity study of the BDV algorithm, the focus was placed on the partitioning heuristics. From these proof sketches, it is clear that the algorithm implementing the balanced heuristic is less complex. The complexity proof sketch is as follows:

Let m = the number of attributes within the original table
Let n = the number of records within the original table.

Case 1: Balanced Heuristic
The algorithm will traverse the entire table <= 2 times. The first time, the algorithm calculates the dissonance values for each attribute: m*n. The second time, the algorithm will check for aggregation: m*n. After every partition, the table will be cut in half. The smaller calculations are dropped.

$\Rightarrow O(m*n)$ with large constants.

Case 2: Unbalanced Heuristic
The algorithm will traverse the entire table <=2 times. The first time, the algorithm calculates the dissonance values for each attribute: m*n. The second time, the algorithm checks for aggregation: m*n. After every partition, the table is reduced by one record (worst case). The first step would be done n-1 times.

$\Rightarrow O(n^2)$ with large constants.

Chapter 6

FUTURE WORK

6.1 Partitioning Heuristics

Future work in partitioning heuristics can include many things. Some common questions and areas that developed naturally through this experiment included different dissonance metrics, tie breaking, and limits placed on partitioning.

Rather than attempting to solve the result of the heuristic, future work could involve developing new partitioning heuristics that did not cause as many ties. This could resolve the problem from the very beginning. In the future, either the dissonance metric or the entire partitioning scheme can be altered. However, future researchers must be careful that the new partitioning scheme, doesn't adversely affect the rest of the algorithm. If the dissonance metric was altered, it may be interesting to find out what type of results would occur if methods from rough sets were implemented.

If the future researcher decides to maintain the current partitioning heuristic, then the tie-breaking scheme can be altered. In this study, tie-breaking was based on a method that employed look ahead. In the future, this type of function could prove to be computationally expensive and may not even be optimal. Therefore, future work can focus on developing a new tie-breaking

scheme that uses some other type of method. In addition, it may be interesting to determine if the number of ties produced for a data set is a direct result of the partitioning heuristic or is dependent upon the data set used. From this study, it was concluded that the number of ties could be related to the partitioning heuristic implemented. Could there be, however, a characteristic within the data set that can be used to judge how many ties may occur?

Finally, toward the end of this study, limitations were placed on the partitioning heuristics. These limitations restricted the partitioning of tables if the dissonance measure did not meet a certain threshold. This opened the door for many speculations. Is there some optimal threshold value that could be used to limit the partitioning activities? Based on the results from the bridges data set, one could justify further research to find some method that would allow the partitioning limit to reflect characteristics of the original table. For example, it could be based on the attribute having the lowest dissonance value. This could possibly improve performance on data sets such as the bridges data set where the lowest dissonance value in the original table was three and the limit was arbitrarily set at two. Finally, in regards to partitioning limits, future research could determine if these limits were domain dependent. For example, in the medical field where it is common to have attributes with three descriptions, a larger partitioning limit might be more appropriate.

6.2 Non-scalar Attributes

The current BDV algorithm does not aggregate or eliminate any non-scalar attributes. There could be many tasks and procedures to deal with this type of

data. The main problem with these non-scalar data types will be in using aggregation to reduce the search space. Current methods that could be used for aggregation include fuzzy, Dempster-Shafer, rough sets, and statistical/probability methods. Future questions about the inclusion of non-scalar attributes within the data set should be focused upon 1) determining where methods of aggregation for non-scalar values will be applied and 2) where within the algorithm it will be done.

The first question could be rephrased into the following: How can the algorithm recognize or differentiate attributes where common groupings for values exist? Moreover, when must a grouping method (fuzzy, rough sets, etc.) be done? One has to keep in mind that this is an unguided approach. Common groupings based on domain knowledge cannot be used. For example, the groupings of high and low blood pressure in the medical field are well known by the numeric values associated with them. Charts will contain numeric values instead of descriptions of high and low. In unguided mining, a method must be developed to achieve the same classification. A general method must be found to deal with all attributes containing non-scalar values.

The second question may have a more direct affect on the organization of the algorithm itself. From the current study, it seems that further study on the placement of these methods will include the application of these methods as a pre-processing step. This would reduce the original data set by the elimination of redundant records. On the other hand, a more interesting study may be done to attempt to find out if there is a certain time within the algorithm that may

produce better results. An optimal solution might be found if a certain type of signal could be developed when a non-scalar attribute was to be grouped.

6.3 Aggregation Methods

From this study, the aggregation method implemented showed serious limitations. Aggregation in this study was done as a final test. Aggregation was done if the following existed:

- All tests for partitioning failed.
- No attribute could be removed based on unique attribute values.
- No attribute could be removed based on a dissonance value of one (single value for all fields).
- The highest dissonance value was between two and the total number of records within the partition.
- Abstraction sets existed for the current attribute.

According to the dissonance values, if a tie existed between two attributes, the attribute with the highest number of redundancies in the resulting table would be aggregated.

One topic for future study could determine if a heuristic existed that could choose the best attribute to aggregate. Could we place a limitation on the dissonance number signaling when an aggregation is to be done? Should the

algorithm be able to aggregate the attribute with the second highest dissonance value rather than the highest value based on the availability of abstraction sets? For example, if no abstraction set existed for the attribute containing the highest dissonance value but one existed for the attribute with the second highest dissonance value, what would happen if we used the attribute with the second highest dissonance value. In other words, could the algorithm be developed to go through a listing of attributes and their dissonance values until the algorithm found an attribute it could abstract? Would the results of this type of approach be significant?

When this algorithm was tested with large data sets, a question that arose was what type of justification was being used to aggregate an attribute. Is there a limit or acceptable number of redundant records that must exist before aggregation is done? For example, if the attribute chosen for aggregation only produces two redundant records, is it sufficient to aggregate the entire attribute based on these two records? When the data set is small, 10 to 20 records, aggregating for two redundant records may be worth it. However, when we enlarge the data set to 1,000 or 2,000 records, do the same two redundant records qualify as a mean for aggregation? It seems that this number may be too small; when the data set is enlarged to such an extent, more redundancies may be required before aggregation is done.

Following the above thought pattern, if we assume aggregation is limited according to the size of the data set, then can the number of redundancies required for aggregation be computed multiple times during run time? Would this make any difference? For example, if we have an original data set of 5,000

records as a starting point, we can limit aggregation to occur only when 300 redundancies can be developed. However, as the algorithm progresses and the number of records within the current table goes below 300, a new limitation will be needed on the number of redundancies. Implementing a function to calculate, at each table, how many redundancies are needed for abstraction could solve this.

6.4 Termination Requirements

This study has shown that in the versions of the sequences that included late aggregation, the final tables in the larger data sets all contained a small number of attributes. It can be inferred from this observation that the termination requirement, while optimal for small data sets, is not optimal for the larger data sets. For further projects, there could be more focus placed on the sizes of the data sets rather than their content. Ideally, researchers would discover a termination requirement that was a function of the data set. This would allow the algorithm to find the best possible termination requirement with little or no background knowledge. In addition to finding optimal termination requirements, it may be of interest to determine if the data set reaches a certain state where the algorithm should be terminated before meeting requirements. Unnecessary work could be eliminated if such a state could be quantified and established.

6.5 Data Sets

The data sets used in this study, while suiting our purposes, are not the only data sets that could have been used. Through this study, it has been shown that the performance of this algorithm is related to the data set that is used. However, a larger study using many more data sets is needed before any general conclusion

can be made. Future work focusing on the data sets should include dependencies between the algorithm and the data sets. Do any exist and, if so, can they be defined, classified, and recognized?

6.6 Methods of Measurement

Throughout this study, it has been a challenge to develop quantitative measurements for determining the quality of the descriptions found during data mining. This area could be a future research project in itself. By developing methods of measurement for these algorithms, bench marking programs could be developed that directly relate to data mining issues rather than only to machine learning or databases.

6.7 Algorithm Development

This study has been a first step toward the development of a data mining technique that uses the blind dissonance value approach. As such, it was intended to be a launching pad for more research in this area. General ideas that can be expanded include the break up and organization of the algorithm's subtasks and the efficiency within these subtasks. If the subtasks are improved or combined in future research, the current results presented in this paper should remain valid.

In addition to improving the existing subtasks, future research must take into account additional data types, namely, scalar data types. The implementation of subtasks to deal with these data types may or may not have an affect on the algorithm.

Finally, the algorithm should be able to accommodate extremely large amounts of data in a reasonable time frame. This issue must be addressed if this algorithm is to have any other future outside the academic world. It was not emphasized because this was only the first step in the development of an algorithm. However, to effectively deal with large data warehouses, potential areas for parallelism and search control must be examined further. An option in this area that does not involve parallelism is to implement this algorithm in a non-iterative manner. This may save some computational time as well as reduce the amount of data that must be made available to the algorithm.

6.8 Results of Algorithm

The BDV algorithm has been designed as an unguided approach to data mining. An area of data mining that is becoming important right now is finding uses for this information in the real world.

In general, the information gathered could be included in some type of decision support system (DSS). If it is included, how does a user determine what relations should be included? A possible answer to this question may be in the development of a new metric. (At this point, the number of instances covered may be used to determine if the relation should be included into a DSS). Another issue is determining what to do when new relations are entered into the system. This is directly related to the integrity of the entire support system.

Another area of study includes the user and the interface to the algorithm. There are two main groups of users for these systems, the technical and the

nontechnical users. The technical user group includes the data analysts, the information manager, and the computer scientist. The nontechnical user would include management, marketing, and other business management users. Future work in this area includes the following:

Developing applications that make use of these results.

Finding a method to display these results that is meaningful to the user.

Developing an interface for the data mining algorithm itself so it can be effectively used by both groups of users.

Chapter 7

CONCLUSION

The content of this study can be used as a starting point for further, more concentrated research in data mining using the blind dissonance approaches. Through this study, a classification of current data mining algorithms has been developed. This classification method has been based on current research concerns: the use of domain knowledge within data mining, and validity and maintenance of the discovered information. Using this background as a starting point, an algorithm has been developed using the least complicated methods and techniques available.

The BDV algorithm does not attempt to find all descriptions within a database. It does not take into account the known dependencies existing within the data set. Depending on the viewpoint - this could be a disadvantage. It can be viewed as a disadvantage because it does not use previous information in the learning process. The results may also be incomplete - eliminating possible known relations where the dependence plays a large role. However, this does not have to be considered a disadvantage. From the viewpoint of unguided data mining, this may be advantageous. It may find other relations when the dependency is either canceled or ignored.

A major concern and issue throughout this study were the metrics used for the analysis of the algorithm. This issue has occurred throughout the history of Artificial Intelligence. It is when a result is based on qualitative measures, such as newness or interestingness, measures that are based on human interaction and

judgment; quantitative measures are always difficult to develop. To continue this study, better metrics for determining success must be developed or an expert in a given field should be employed to verify the results.

Data mining is a very "hot" area right now. Challenges facing today's researchers include the volume of data coming in, the ability of the algorithm to perform efficiently, the success rate of these algorithms, and the translation of the discovered information into something useful. The blind dissonance approach fits into the larger picture by offering a new method based on simple techniques and minimum domain knowledge. If this method proves successful, it will offer the user the ability to push a button and wait for an answer, since very little user interaction is required for algorithm. It would be available for end users knowing little or almost nothing about the data mining procedure.

Bibliography

[Agrawal 1992] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. *An Interval Classifier for Database Mining Applications*, Proceedings of the 18th VLDB Conference, 1992. pp. 560-573.

[Agrawal 1993] R. Agrawal, T. Imielinski, and A. Swami. (1993) *Database Mining: A Performance Perspective*, IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, Dec. 1993. Pp. 914-925.

[Anwar 1992] T. Anwar, H. Beck, S. Navathe. *Knowledge Mining by Imprecise Querying: A Classification-Based Approach* IEEE 1992. pp.622-630.

[Bell 1993] D. A. Bell. *From Data Properties to Evidence*, IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, Dec. 1993. pp. 965-968.

[Bergadano 1993] F. Bergadano, *Inductive Database Relations*, IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, Dec. 1993. pp. 969-972.

[Brodie 1992] M. L. Brodie and J. Mylopoulos *Artificial Intelligence and databases: Dawn, midday or sunset?*, Canadian Information Processing July/Aug. 1992. pp. 25-27.

[Chiu 1991] D. K. Y. Chiu, A. K. C. Wong, and B. Cheung. *Information Discovery through Hierarchical Maximum Entropy Discretezation and Synthesis* in Knowledge Discovery in Databases, pp. 125-141.

[Clark, Niblett 1989] P. Clark, and T. Niblett *The CN2 Induction Algorithm*, Machine Learning, vol. 3, 1989. pp. 261-283.

[Davidson 1993] C. Davidson. *What your Database Hides Away*, New Scientist, vol. 9, Jan 1993. pp. 28-31.

[Dhar, Tuzhilin 1993] V. Dhar and A. Tuzhilin, *Abstract-Driven Pattern Discovery in Databases*, IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, Dec 1993. pp. 926-938.

[Dietterich 1983] T. Dietterich and R. Michalski. *A Comparative Review of Selected Methods for Learning From Examples*, in Machine Learning: An Artificial Intelligence Approach, 1983. pp. 41-83.

[Drastal, Czako 1992] G. Drastal, G. Czako, and S. Raatz. *Induction in an Abstraction Space: A Form of Constructive Induction*, in Machine Learning, 1992. pp. 702-711.

[Duran, Odell 1974] B. Duran and P. Odell. *Cluster Analysis: A Survey* Lecture Notes in Economics and Mathematical Systems, vol. 100, 1974.

[Dzeroski, Lavrac 1993] S. Dzeroski and N. Lavrac. *Inductive Learning in Deductive Databases*, IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, Dec 1993. pp. 939-949.

[Fayyad, Simoudis 1995] U. Fayyad and E. Simoudis. *Knowledge Discovery in Databases,* Tutorial MA1 IJCAI-95.

[Fisher 1987] D. Fisher. *Knowledge Acquisition Via Incremental Conceptual Clustering,*" in Machine Learning, vol. 2, 1987. pp. 140-172.

[Fisher, Langley 1985] D. Fisher and P. Langley. *Approaches to Conceptual Clustering,* in Proceedings of the Ninth International Joint Conference on Artificial Intelligence 1985, pp. 691-697.

[Frawley 1991] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. *Knowledge Discovery in Databases: An Overview*, in Knowledge Discovery in Databases. pp. 1-31.

[Gallare 1984] H. Gallare, J. Minker and J. Nicolas. *Logic and Databases: A Deductive Approach,* Computing Surveys, vol. 16, no. 2, June 1984, pp. 53-185.

[Han 1993] J. Han, Y. Cai and N. Cercone. *Data-Driven Discovery of Quantitative Rules in Relational Databases*, IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, Dec 1993. pp. 29-40.

[Han 1992] J. Han, Y. Cai, and N. Cercone. *Knowledge Discovery in Databases: An Attribute-Oriented Approach*, in Proceedings of the 18th Very Large Database Conference; Vancouver, B. C. Canada, 1992. pp. 547-559.

[Hayes-Roth 1978] F. Hayes-Roth and J. McDermott. *An Interference Matching Technique for Inducing Abstractions*, Communications of the ACM, vol. 21, no. 5, May 1978. pp. 401-410.

[Holsheimer, Siebes 1994] M. Holsheimer and A. Siebes. *Data Mining: The Search for Knowledge in Databases*, CWI CS-R9406, 1994.

[Iba 1988] W. Iba, J. Wogulis, and P. Langley. *Trading off Simplicity and Coverage in Incremental Concept Learning* in Proceedings of the Fifth International Conference on Machine Learning, 1988. pp. 73-79.

[Imielinski, Lipski 1984] T. Imielinski and W. Lipski. *Incomplete Information in Relational Databases*, in Readings in Artificial Intelligence and Databases, Eds. Mylopolous and Brodie, 1984. pp. 342-360.

[Knoblock 1990] C. Knoblock. *Learning Abstraction Hierarchies for Problem Solving*, Machine Learning, 1990. pp.923-928.

[Lewinson 1994] L. Lewinson *Data Mining: Tapping into the Mother Lode,* Database Programming & Design, Feb. 1994. pp. 50-56.

[Langley 1981] P. Langley, G. Bradshaw and H. Simon. *BACON.5: The Discovery of Conservation Laws,* in Proceedings of the Seventh International Joint Conference on Artificial Intelligence, 1981, pp. 121-126.

[Langley 1996] P. Langley. Elements of Machine Learning, Morgan Kaufmann Publishers, 1996.

[Lenat, Guha 1990] D. Lenat and R. V. Guha. Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project. Addison-Wesley, 1990.

[Manago, Kodratoff 1987] M. V. Manago and Y. Kodratoff. *Noise and Knowledge Acquisition*, Proceedings of the Tenth International Joint Conference of Artificial Intelligence, 1987, pp. 348-354.

[Matheus 1993] C. J. Matheus, P. K. Chan and G. Piatetsky-Shapiro. *Systems for Knowledge Discovery in Databases*, IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, Dec 1993. Pp. 903-913.

[Merz, Murphy 1996] C. J. Merz and P. M. Murphy. UCI Repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science. 1996.

[Michalski 1993] R. Michalski, *A Theory and Methodology of Inductive Learning*, in Machine Learning: An Artificial Intelligence Approach, ed. Michalski, Carbonell, Mitchell, 1993, pp. 83-134.

[Michalski 1990] R. Michalski and Y. Kodratoff. *Research in Machine Learning: Recent Progress, Classification of Methods, and Future Directions*, in Machine Learning, an Artificial Approach, Vol. 3, Eds. Kodratoff and Michalski. 1990, pp. 3-30.

[Mitchell 1982] T. M. Mitchell. *Generalization as Search*, Artificial Intelligence, vol. 18, 1982, p.203-226.

[Pal 1992] N. Pal, J. Bezdek and R. Hemasinha. *Uncertainty Measures for Evidential Reasoning I: A Review*, International Journal of Approximate Reasoning, vol. 7, Dec 1992, pp. 165-183.

[Pal 1993] N. Pal, J. Bezdek and R. Hemasinha. *Uncertainty Measures for Evidential Reasoning II: A New Measure of Total Uncertainty*, International Journal of Approximate Reasoning, vol. 8, January 1993, pp.1-15.

[Pawlak 1981] Z. Pawlak. Rough sets: Theoretical Aspects of Reasoning About Data, Kluwer Academic publishers, 1991.

[Pazzani, Kibler 1992] M. Pazzani and D. Kibler. *The Utility of Knowledge in Inductive Learning*, Machine Learning, vol. 9, 1992, pp. 57-94.

[Piatetsky-Shapiro, Frawley 1991] G. Piatetsky-Shapiro and W. Frawley (eds.). Knowledge Discovery in Databases, AAAI Press 1991.

[Piatetsky-Shapiro, Matheus 1992] G. Piatetsky-Shapiro and C. Matheus. *Knowledge Discovery Workbench for Exploring Business Databases*, International Journal of Intelligent Systems, vol. 7, 1992, pp. 675-686.

[Quinlan 1986] J. R. Quinlan. *Induction of Decision Trees*, Machine Learning, vol. 1, 1986, p.81-106.

[Quinlan2 1986] J. R. Quinlan. *Unknown Attribute Values in Induction*, in Proceedings of Sixth International Workshop on Machine Learning, 1986, pp. 164-168.

[Ragavan 1993] H. Ragavan, L. Rendell, M. Shaw, A. Tessmer. *Complex Concept Acquisition through Directed Search and Feature Caching*, in Proceedings of the 13th IJCAI, 1993 pp. 946-951.

[Reich 1989] Yoram Reich and Steven J. Fenves. *Incremental Learning for Capturing Design Expertise*. Technical Report: EDRC 12-34-89, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA. 1989.

[Rendell 1993] L. Rendell and H. Ragavan. *Improving the Design of Induction Methods by Analyzing Algorithm Functionality and Data-Based Concept Complexity*, in Proceedings of the 13th IJCAI 1993, pp. 952-958.

[Schlimmer 1987] J. S. Schlimmer. *Concept Acquisition through Representational Adjustment (Technical Report 87-19).* Doctoral dissertation, Department of Information and Computer Science, University of California, Irvine. 1987.

[Shen 1991] W. Shen. *Discovering Regularities from Large Knowledge Bases*, in Eighth International Workshop of Machine Learning, 1991, pp. 539-543.

[Smyth, Goodman 1992] P. Smyth and R. Goodman. *An Information Theoretical Approach to Rule Induction from Databases*, IEEE Transactions on Knowledge and Data Engineering, vol. 4, no. 4, August 1992. pp. 301-316.

[Stonebraker 1993] M. Stonebraker, R. Agrawal, U. Dayal, E. Neuhold, A. Reuter. *DBMS Research at a Crossroads: The Vienna Update*, in Proceedings- VLBD, 1993, pp. 688-692.

[Ullman 1988] J. Ullman. Principles of Database and Knowledge-Based Systems. Vol. 1, Computer Science Press. 1988.

[Walker 1980] A. Walker. *On Retrieval from a Small Version of a Large Data Base*, in Proceedings VLDB, 1980, pp. 40-54.

[Yoon, Kerschberg 1993] J. P. Yoon and L. Kerschberg. *A Framework for Knowledge Discovery and Evolution in Databases*, IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, 1993, pp. 973-979.

[Zhong, Ohsuga 1993] N. Zhong and S. Ohsuga. *A Decomposition Based Induction Model for Discovering Concept Clusters from Databases*, in Algorithmic Learning Theory Proceedings, ALT (Eds.) Jantke, Kobayashi, Tomita, and Yokomori. 1993, pp. 384-397.

[Ziarko 1991] W. Ziarko. *The Discovery, Analysis, and Representation of Data Dependencies in Databases*, in Knowledge Discovery in Databases, pp. 195-210.

[Ziarko, Shan 1995] W. Ziarko and N. Shan. *Knowledge Discovery as a Search for Classification*, CSC Workshop on Rough Sets at ACM 1995.

[Zytkow 1993] J. Zytkow and R. Zembowicz. *Database Exploration in Search of Regularities*, Journal of Intelligent Information Systems, vol. 2, 1993. pp. 39-81.

[Zytkow Baker, 1991] J. Zytkow and J. Baker. *Iterative Mining of Regularities in Databases* in Knowledge Discovery in Databases, 1991, pp. 31-55.

Appendix A - Metric Definitions & Applications

A.1 Partitions

To study the partitioning habits of the algorithm, the number of partitions and a trace of the partitions were collected. The term refers to the count of the number of sub-tables created from the original table to the final table. Forming a partition within this system is work. It was believed that a low number of partitions was desirable because it would show that with a small amount of work suitable results can be created. Using this metric, conclusions would include the performance of the partitioning heuristic, and the sequencing of the aggregation and partitioning tasks within the algorithm.

A.2 Partitioning Traces

Partitioning traces are diagrams representing the actions the algorithm has taken. From these diagrams, quantitative measures of depth and breadth are taken. These measures are used for comparison of heuristics and sequencing of subtasks. The depth of the trace represents the partitions formed to reach the final table. The breadth of the trace represents the number of ties unsuccessfully resolved by the partitioning heuristic. If a tie is not resolved among attributes after the partitioning heuristic has been applied, two possible results from these ties have been enumerated.

The partitioning traces are to be read from top to bottom. The top of the trace represents the original table. Each edge of the table represents either a single partition, or a group of partitions. The groupings occur when there are no ties due to the heuristic. The edges are labeled according to the values from the

partitioning attributes used to form the new table; each value will represent one partition. The bottom nodes of the trace represent the number of instances covered by the final tables.

For example, given a partition of the music album data set, (Table A.2.1) a tie occurs between two attributes containing the minimum dissonance when the balanced heuristic is used: On *Tour?* and *Citizen*. Because a tie has occurred, the partitions from both attributes will be enumerated and followed until the termination requirement is met. The partitioning trace incorporates these ties and their enumeration by creating two edges within the diagram. One edge represents the first tied attribute, while the second edge represents the other attribute. See Diagram A.2.1.

Group	*Type*	*Record Label*	*Release Date*	*Citizen*	*On Tour*
REM	Rock/Pop	Warner Bros.	Nov1994	USA	yes
U2	Rock/Pop	Island	Apr1993	Foreign	no
REM	Rock/Pop	Warner Bros.	Jun1990	USA	yes
Nirvana	Rock/Pop	Geffen	Sept1989	USA	no
Nirvana	Rock/Pop	Geffen	Nov1994	USA	no
The Stone Roses	Alternative	Silvertone	Apr1995	Foreign	no
Pearl Jam	Rock/Pop	Epic	1991	USA	yes
Sting	Rock/Pop	A&M	Apr1993	Foreign	no
The Cranberries	Alternative	Island	Apr1995	Foreign	yes
Pearl Jam	Rock/Pop	Epic	Nov1992	USA	yes
James	Alternative	Mercury	Sept1993	Foreign	yes
Squeeze	Alternative	A&M	1987	Foreign	no

Table A.2.1 - Music album data set.

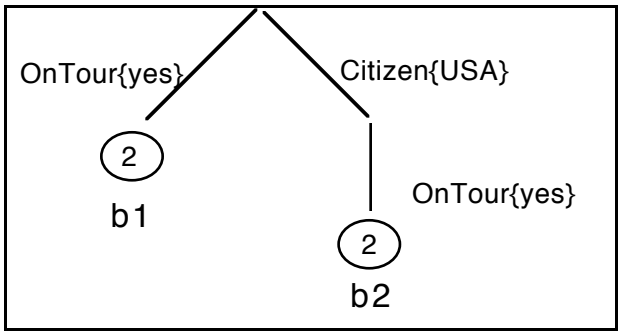

Diagram A.2.1 - Partitioning Trace based on music album data set.

Traces are used to form a better idea of how these partitions are made, to test the partitioning heuristics, and to enumerate the differences among the versions. The trace aid in describing the partitioning patterns found within the algorithm's execution. This information is used in the study of the performance of the partitioning heuristics - specifically showing the number of ties occurring after the partitioning heuristic has been completed. It was hoped that these traces might produce a different view toward further tie breaking requirements. A partitioning heuristic creating a small number of ties, resulting in a narrow trace is preferable.

A.3 Aggregations & Dependencies

The number of dependencies found within an execution of the algorithm and the number of aggregations carried out has been collected for analysis. These two metrics were found to be highly correlated and when they were in a certain range - highly desirable. These metrics were found to be directly affected by the sequencing of tasks within the algorithm. These metrics could lead to the development of the benefits of one task sequencing within the algorithm over another and the advantages or disadvantages of the implementation of the partitioning task.

In addition, since dependencies are considered informative, if any are found through this algorithm, it is a way of comprehending and judging the results of the different heuristics. The number of aggregations done results in learning how quickly the search space is limited.

A.4 Final Table Characteristics

This area of results is qualitative; no measures were available to determine if a result was good or bad. Therefore, the following quantitative measures have been applied. The final table represents the discovered relation within this algorithm. The contents of this table have been studied according to the number of attributes, the number of records covered, and the percentage of records covered in the original data set.

These numbers can be used to determine potential uses for the results. The number of attributes within the final table directly corresponds to the semantic complexity of the final relationship. The number of records covered and percentage covered allows the user to determine potential applications of the relation. Also, the table characteristics could be used to signal a change in execution - if the heuristic involved had been changed or if the sequencing of aggregation and partitioning subtasks was in question it's affects might be visible in these characteristics.

A.5 Collected Information

The results of this algorithm are in the form of rules: antecedent and consequences. The contents of the collected information gathered throughout this algorithm are used as the antecedents for the final table (representing the

consequences). The contents of the collected information are affected by the carrying out of aggregation using the captured abstraction sets. Collected information is a listing of attributes with a certain value associated with them. It was stored according to high information content. Useful information is gathered through other methods rather than aggregation and can be used to produce the same consequences found in the final table.

The contents of these two types of information were studied and applied to questions dealing with the utility of identifying and capturing abstraction sets. In addition, if the contents of these results were different according to the partitioning heuristic used or the sequencing implemented, the results were used to form conclusions on these other topics.

B.1 Partitioning

The number of partitions created during execution was used to determine the amount of work done within the algorithm. This number was collected for each heuristic and compared over the different sequencing of subtasks. Overall, the balanced heuristic performed the best, generating the least amount of partitioning tables. However, to form specific conclusions about the unbalanced and arbitrary heuristics the results from each version must be studied. In some cases, the performances of the arbitrary and unbalanced heuristic showed similarities, at some points, the arbitrary heuristic out performed the unbalanced heuristic.

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	2	2.5	2.25
Album	1.5	3	1.5
Bridges	3	7	5.6
Mushroom 500	6	16	?
Mushroom 1,000	7	11.75	?
Mushroom 3,000	8	14.5	?
EPA 2,000	3	5	4.6
EPA 3,000	4	5	4.6

Table B.1.1 Late Aggregation, Unlimited Partitioning - number of partitions created

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	2	2.5	2
Album	1	2	0.5
Bridges	1	1	1
Mushroom 500	1	3.16	3.12
Mushroom 1,000	2.3	2	2.4
EPA 2,000	2	2	2
EPA 3,000	2	2	2

Table B.1.2 Early Aggregation, Unlimited/Limited Partitioning - number of partitions created

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	2	2.5	2.25
Album	1.5	3	1.5
Bridges	0	0	0
Mushroom 500	6	16.16	?
Mushroom 1,000	7	11.75	?
EPA 2,000	3	4	3.6
EPA 3,000	3	4	3.6

Table B.1.3 Late Aggregation, Limited Partitioning - number of partitions created

The question marks within the tables have been added for the occasion when the algorithm did not finish under a reasonable amount of time.

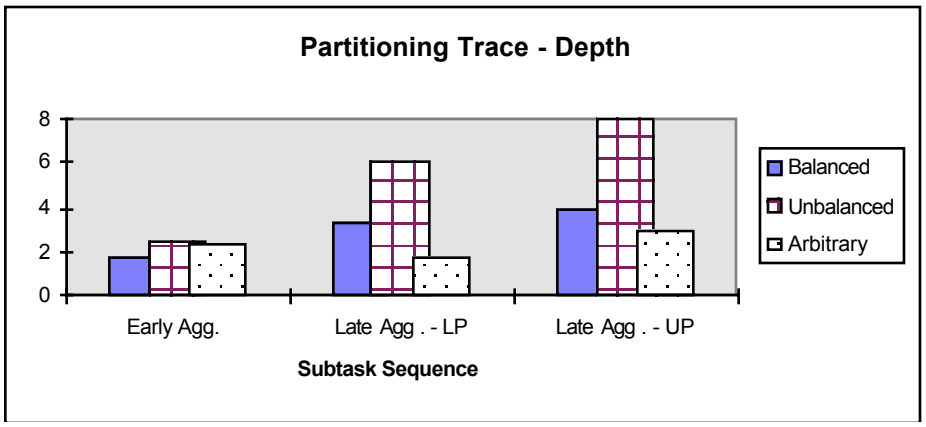

Diagram B.1.1 Partition trace comparison subtask sequence vs. levels within the tree.

The balanced heuristic created the smallest number of ties during execution, therefore, having narrow partitioning traces. As far as depth is concerned, the control traces were generated by the arbitrary heuristics. If either the balanced or the unbalanced heuristic traces maintain a shorter path from the origin to the final node than the path generated by the arbitrary heuristic, then the heuristic used was beneficial to the system. The depth of the partitioning trace is directly

related to the number of partitions created and therefore the amount of work the algorithm is performing as well.

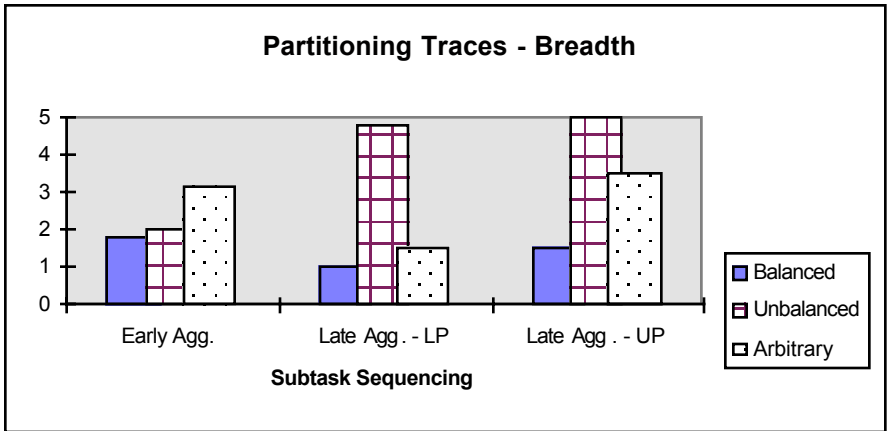

Diagram B.1.2 Partition trace comparison subtask sequencing vs. breadth of trace.

	Data Sets	*Late Aggregation*	*Early Aggregation*
DEPTH	Mushroom 500	10	2.5
	Mushroom 1,000	13.5	2.5
	EPA 3,000	3.7	2
	----------	----------	----------
BREADTH	Mushroom 500	6.5	2.5
	Mushroom 1,000	5.5	2
	EPA 3,000	1.5	2

Table B.1.4 Comparison of Depth and Breadth in larger data sets.

Through the partitioning traces, the affect of the subtask sequencing is shown not only in the physical diagram, but also in the number of records covered by the trace (shown in the leaf nodes of the trace). The versions containing Early aggregation can be used to form a minimum limit. The versions containing late partitioning and unlimited partitioning form a maximum limit. The traces falling in between these two limits were generated with late aggregation and limited partitioning. This middle ground represents the ideal results.

B.2 Aggregations & Dependencies

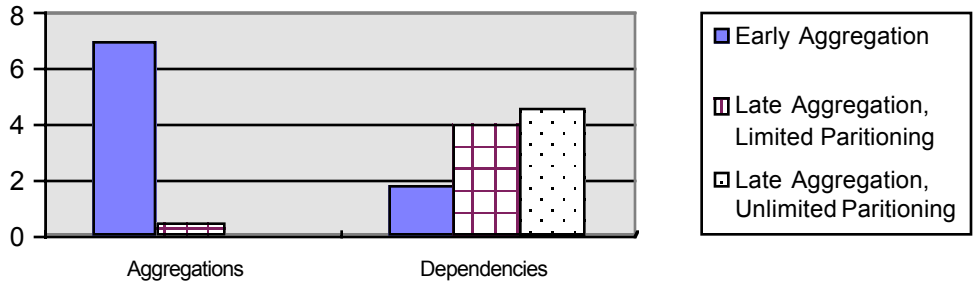

Diagram B.2.1 Number of Aggregations and Dependency Comparison

Data was collected for abstraction sets, but no aggregation was done when the partitioning subtask was done first.

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	0	0	0
Album	0.5	0	0.5
Bridges	0	1.33	1.5
Mushroom 500	5	10	?
Mushroom 1,000	14	8.4	?
Mushroom 3,000	14	7	?
EPA 2,000	10	9	9.33
EPA 3,000	10	9	9.33

Table B.2.1 Dependencies Found in Late Aggregation, Unlimited Partitioning

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	1.5	3	1.5
Album	1	2	1
Bridges	9	9	9
Mushroom 500	11	11	4.6
Mushroom 1,000	12	12	12
EPA 2,000	7	7	7
EPA 3,000	7	7	7

Table B.2.2 Aggregations performed in Early Aggregation and Unlimited/Limited Partitioning

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	0	0	0
Album	0.5	0	0.5
Bridges	0	0	0
Mushroom 500	4	5	4.6
Mushroom 1,000	4.3	4	4.4
EPA 2,000	2	2	2
EPA 3,000	2	2	2

Table B.2.3 Dependencies found in Early Aggregation and Unlimited/Limited Partitioning

The late aggregation with limited partitioning differed from the late aggregation and unlimited partitioning version only in the bridges and EPA data sets. The bridges data set is not applicable to this discussion because aggregation was not done and no dependencies were discovered. The number of aggregations done with limited partitioning was seven across all EPA data sets. Across all data sets, the number of dependencies found was ten.

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
EPA (all sets)	10	10	10

Table B.2.4 Dependencies found in late aggregation and limited partitioning

B.3 Final Table Attributes

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	4	4	3.25
Album	2	2	2
Bridges	4	2	2.75
Mushroom 500	2	1.5	?
Mushroom 1,000	1	1	?
Mushroom 3,000	1	1	?
EPA 2,000	2	2	2
EPA 3,000	2	2	2

Table B.3.1 Attributes within Final Tables found in Late Aggregation and Unlimited Partitioning

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	2.5	1.5	2
Album	2	1	2
Bridges	1	1	1
Mushroom 500	3	3.33	3.375
Mushroom 1,000	3	1	3.4
EPA 2,000	3.5	3.5	3.5
EPA 3,000	3.5	3.5	3.5

Table B.3.2 Attributes within Final Tables found in Early Aggregation and Unlimited Partitioning

The late aggregation with limited partitioning differed from the late aggregation with unlimited partitioning version only in the large EPA data sets. The EPA final tables contained a single field for all heuristics. This supports the theory that as the data sets grow in size, the number of attributes within the final table shrinks.

B.4 Percentage Covered

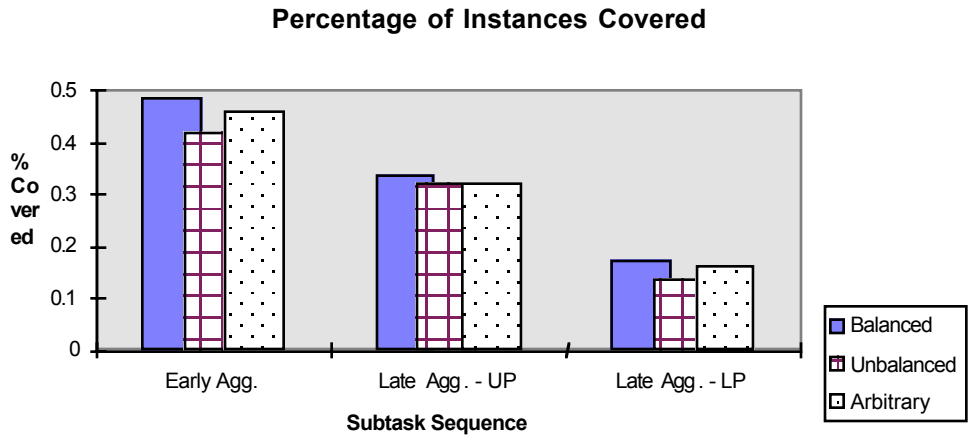

Diagram B.4.1 Percentages of Instances covered.

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	37%	31%	31%
Album	62.5%	50%	62.5%
Bridges	3.65%	3.17%	4.10%
Mushroom 500	0.6%	0.43%	?
Mushroom 1,000	0.3%	0.3%	?
EPA 2,000	0.067%	0.1%	0.067%
EPA 3,000	0.067%	0.1%	0.067%

Table B.4.1 - Percentage of final records covered by final tables found with late aggregation and limited partitioning

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	35%	32%	32%
Album	75%	61%	75%
Bridges	28%	80%	55%
Mushroom 500	80%	65%	60%
Mushroom 1,000	78%	12%	58%

Table B.4.2 - Percentage of final records covered by final tables found with early aggregation and unlimited/limited partitioning

Data Set	*Balanced*	*Unbalanced*	*Arbitrary*
University	37%	31%	31%
Album	62.5%	62.5%	62.5%
Bridges	100%	100%	100%
Mushroom 500	0.6%	0.43%	?
Mushroom 1,000	0.3%	0.3%	?
EPA 2,000	0.33%	0.33%	0.33%
EPA 3,000	0.33%	0.33%	0.33%

Table B.4.3 - Percentage of final records covered by final tables found with late aggregation and limited partitioning

B.5 Collected and Useful Information

	Early Aggregation Unlimited/Limited Partitioning		Late Aggregation Unlimited Partitioning		Limited Partitioning	
Data Set	*Collected Information*	*Useful Information*	*Collected Information*	*Useful Information*	*Collected Information*	*Useful Information*
University	3.5	2	2.5	2	2.5	2
Album	2	2	1	1	2	2
Bridges	10	2	7	7	9	2
Mushroom 500	20	9	21	21	21	21
Mushroom 1,000	20	8	22	22	22	22
EPA 2,000	19	6.5	14	14	19	15
EPA 3,000	19	6.5	14	14	15	13

Table B.5.1 Collected and Useful Information - Balanced Heuristic, Early and Late Aggregation sequences.

C.1 Balanced Heuristic, Late Aggregation, Unlimited Partitioning

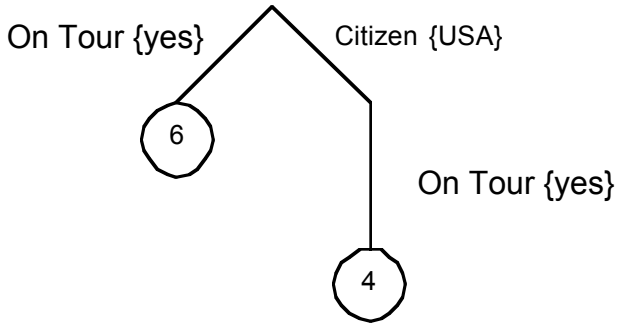

Diagram C.1.1: Music Album Data Set

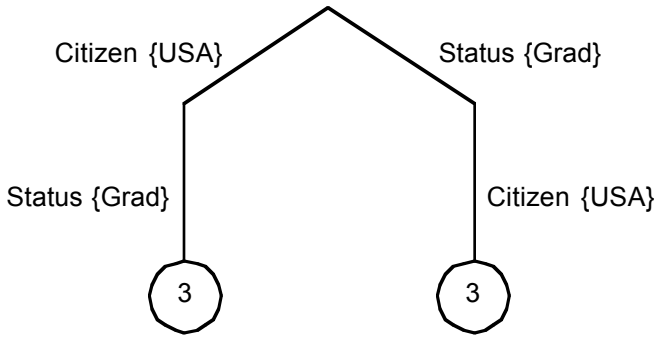

Diagram C.1.2: University Data Set

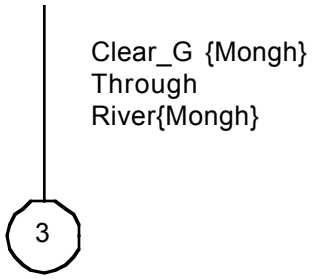

Diagram C.1.3: Bridges Data Set

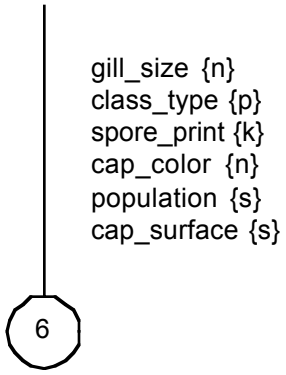

Diagram C.1.4: Mushroom 500 Data Set

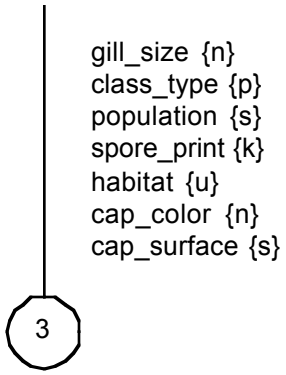

Diagram C.1.5: Mushroom 1,000 Data Set

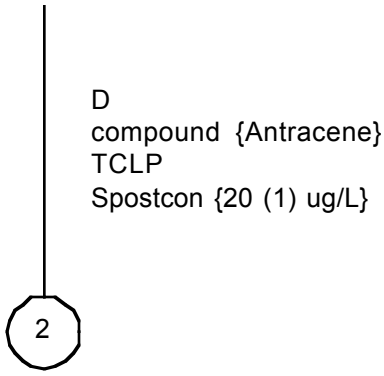

Diagram C.1.6: EPA 3,000 Data Set

C.2 Balanced Heuristic, Late Aggregation, Limited Partitioning

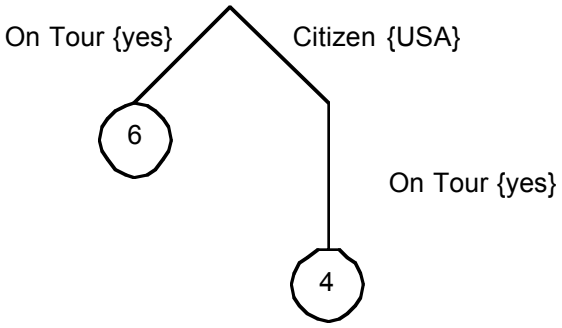

Diagram C.2.1: Music Album Data Set

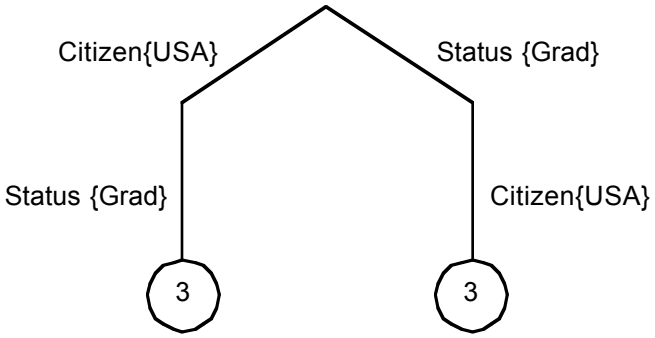

Diagram C.2.2: University Data Set

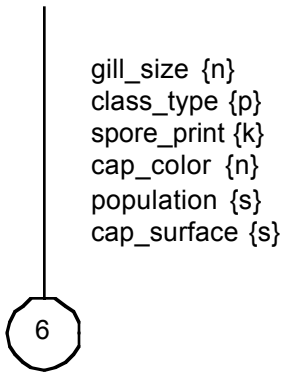

Diagram C.2.3: Mushroom 500 Data Set

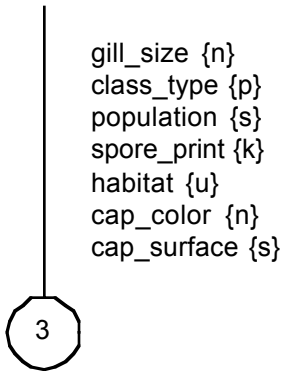

Diagram C.2.4: Mushroom 1,000 Data Set

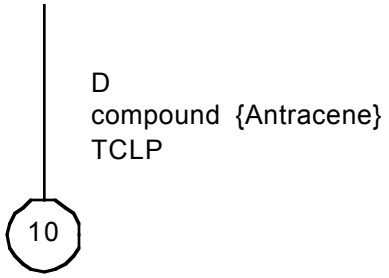

Diagram C.2.5: EPA 3,000 Data Set

C.3 Unbalanced Heuristic, Late Aggregation, Unlimited Partitioning

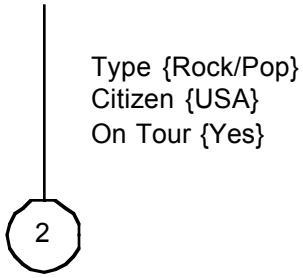

Diagram C.3.1: Music Album Data Set

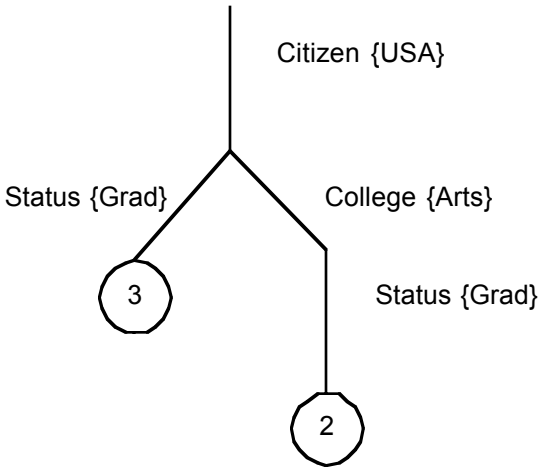

Diagram C.3.2: University Data Set

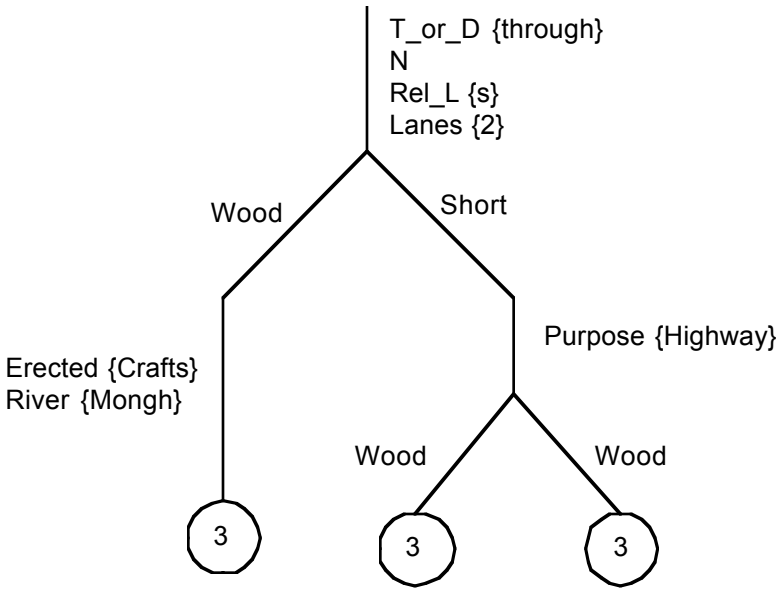

Diagram C.3.3: Bridges Data Set

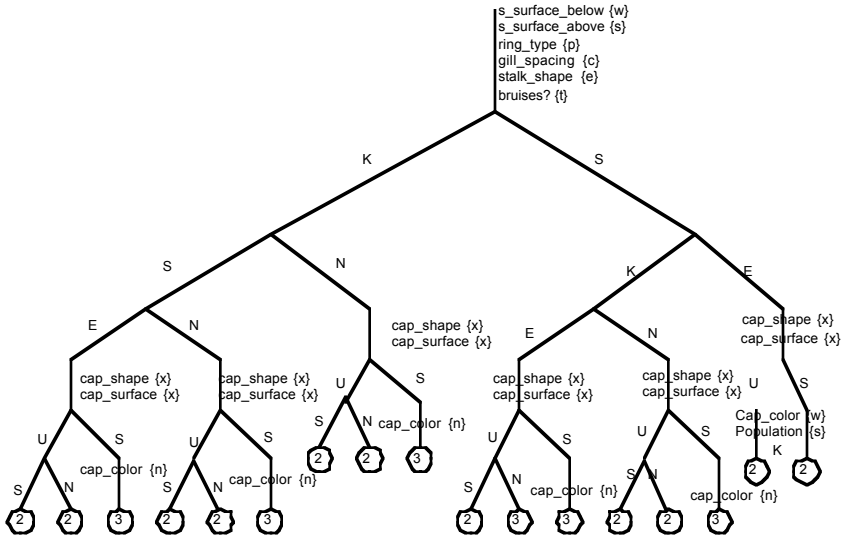

Diagram C.3.4: Mushroom 500 Data Set

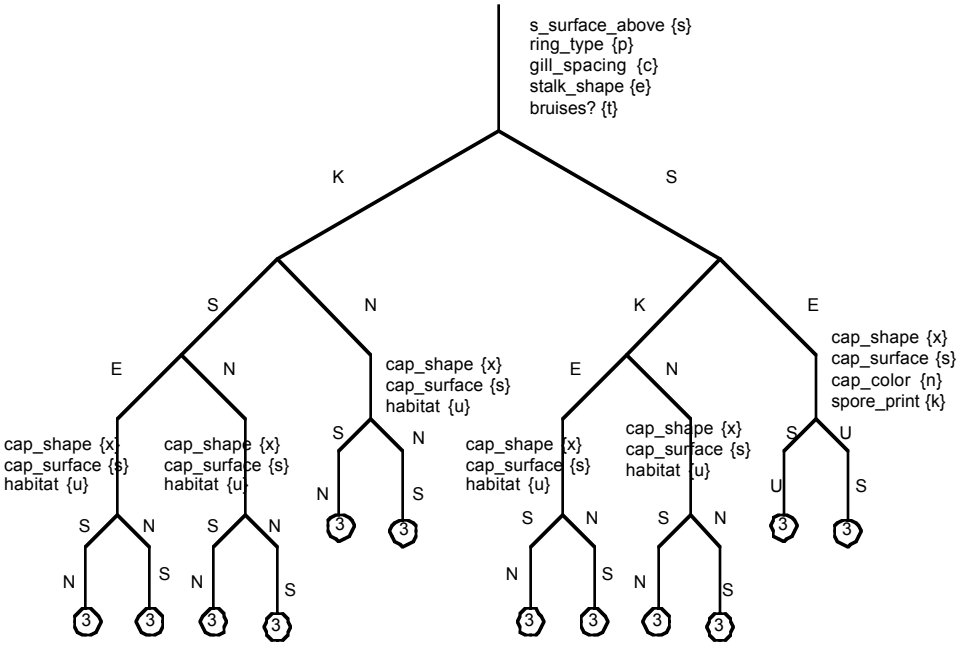

Diagram C.3.5: Mushroom 1,000 Data Set

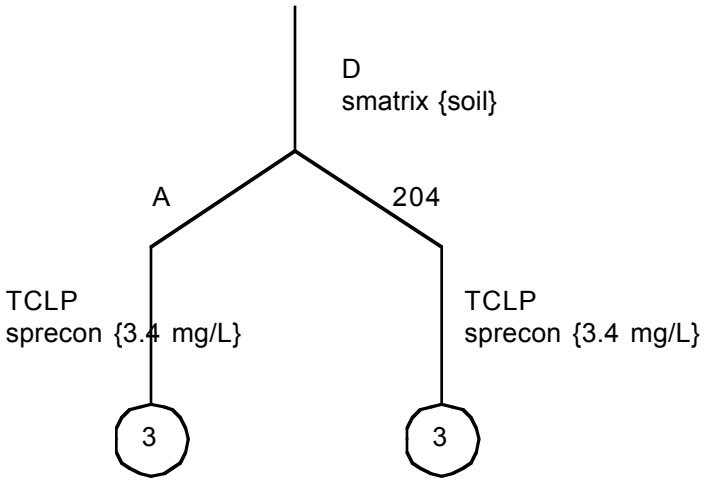

Diagram C.3.6: EPA 3,000 Data Set

C.4 Unbalanced Heuristic, Late Aggregation, Limited Partitioning

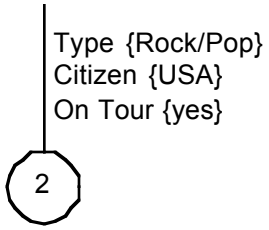

Diagram C.4.1: Music Album Data Set

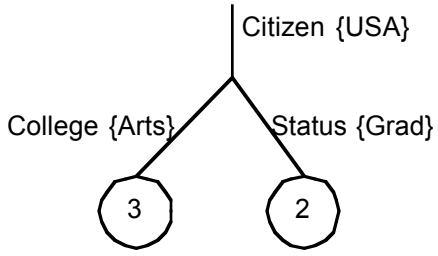

Diagram C.4.2: University Data Set

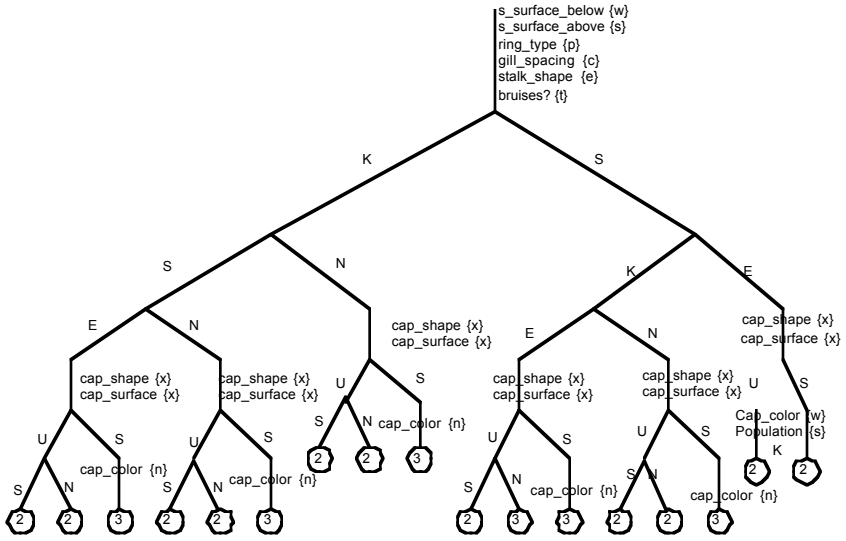

Diagram C.4.3: Mushroom 500 Data Set

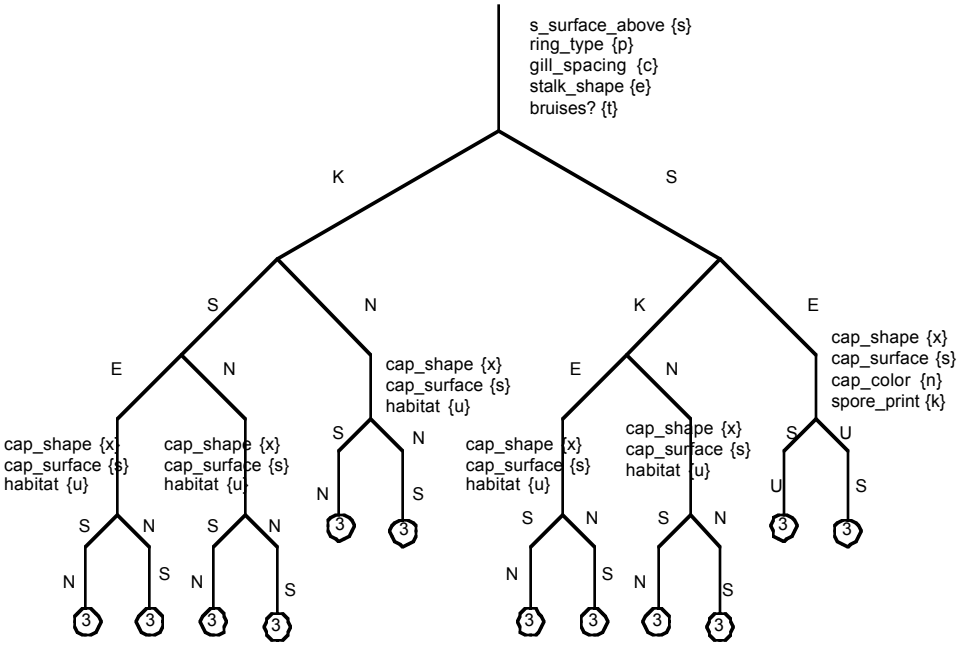

Diagram C.4.3: Mushroom 1,000 Data Set

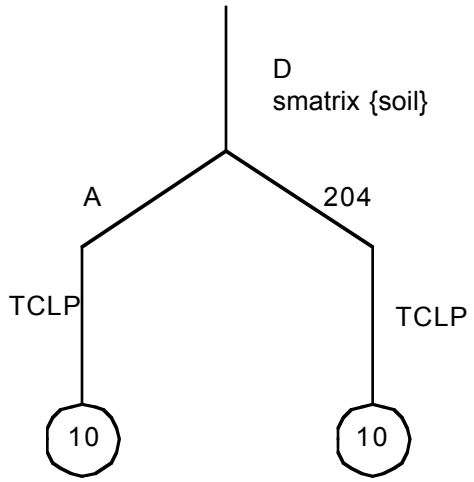

Diagram C.4.4: EPA 3,000 Data Set

C.5 Arbitrary Heuristic, Late Aggregation, Unlimited Partitioning

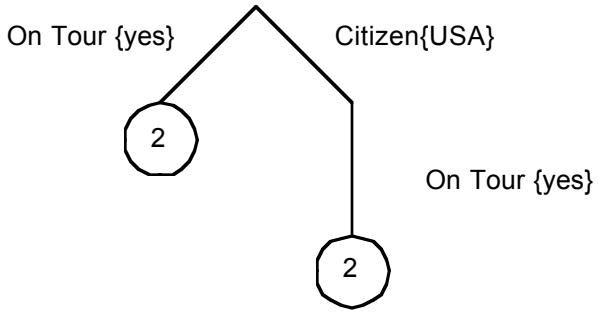

Diagram C.5.1: Music Album Data Set

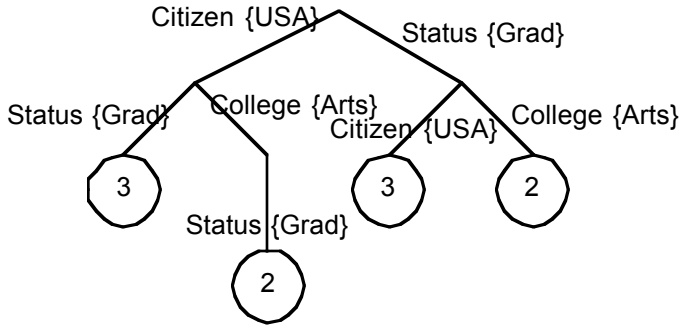

Diagram C.5.2: University Data Set

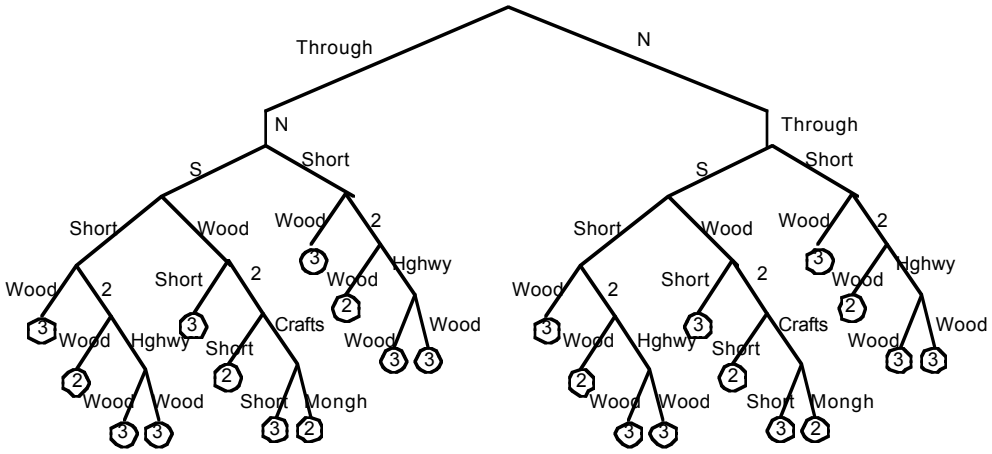

Diagram C.5.3: Bridges Data Set

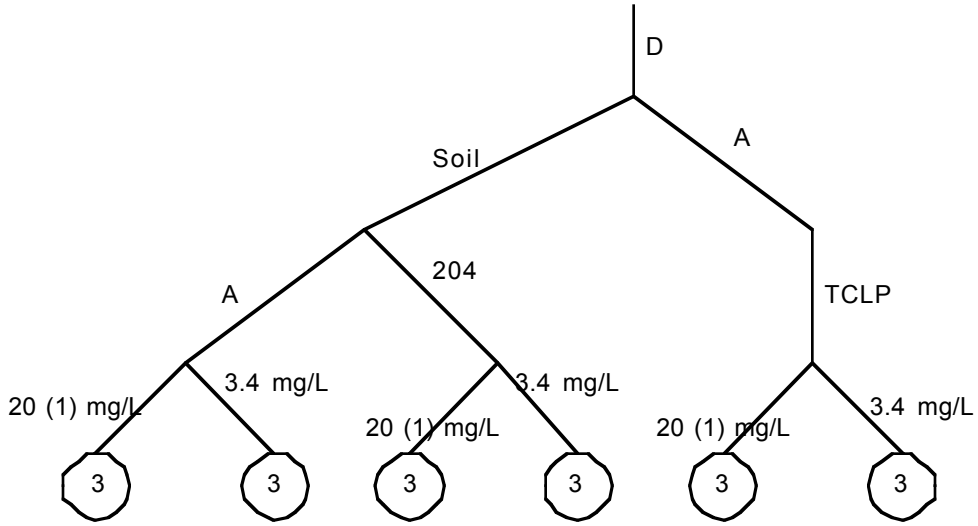

Diagram C.5.4: EPA 3,000 Data Set

C.6. Arbitrary Heuristic, Late Aggregation, Limited Partitioning

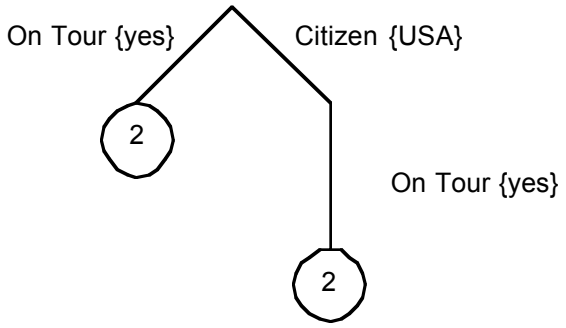

Diagram C.6.1: Music Album Data Set

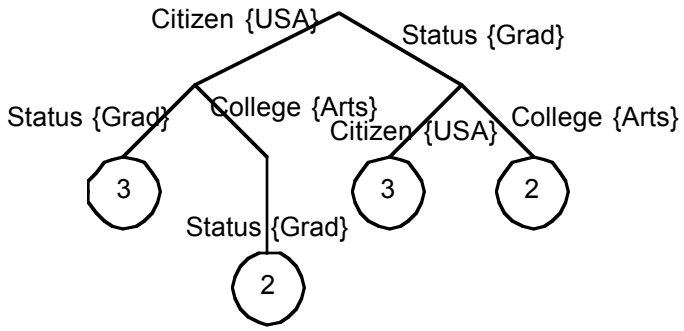

Diagram C.6.2: University Data Set

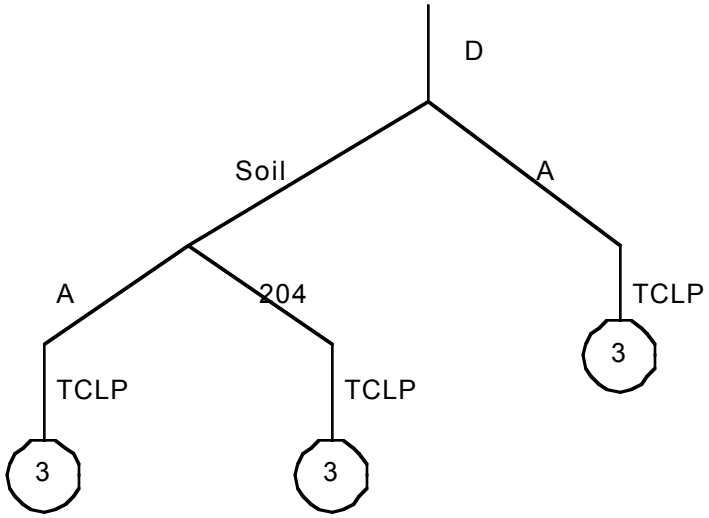

Diagram C.6.3: EPA 3,000 Data Set

C.7. Balanced Heuristic, Early Aggregation, Limited and Unlimited Partitioning

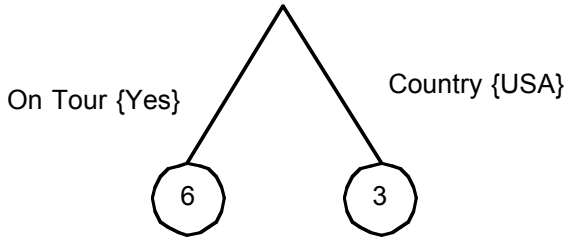

Diagram C.7.1: Music Album Data Set

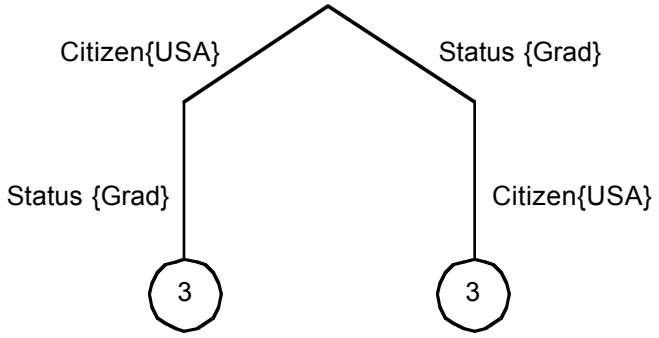

Diagram C.7.2: University Data Set

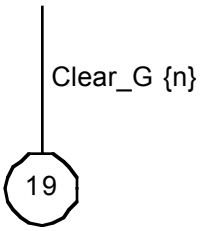

Diagram C.7.3: Bridges Data Set

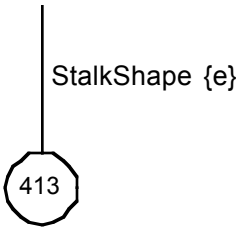

Diagram C.7.4: Mushroom 500 Data Set

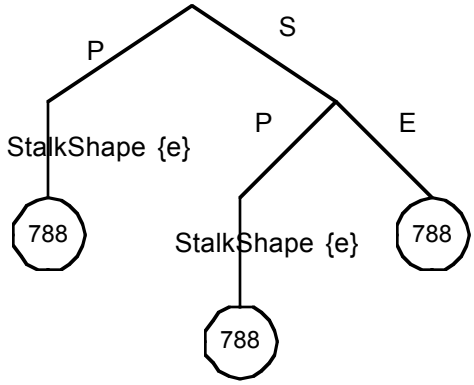

Diagram C.7.5: Mushroom 1,000 Data Set

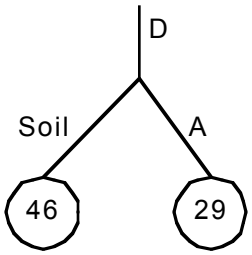

Diagram C.7.6: EPA 3,000 Data Set