EFFICIENT GENERATION OF REDUCTS AND DISCERNS FOR CLASSIFICATION

A thesis presented to

the faculty of

the Russ College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

James T. Graham

June 2007

This thesis entitled

EFFICIENT GENERATION OF REDUCTS AND DISCERNS FOR CLASSIFICATION

by

JAMES T. GRAHAM

has been approved for

the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology by

Janusz A. Starzyk

Professor of Electrical Engineering and Computer Science

Dennis Irwin

Dean, Russ College of Engineering and Technology

Abstract

GRAHAM, JAMES T., M.S., June 2007, Electrical Engineering and Computer Science

EFFICIENT GENERATION OF REDUCTS AND DISCERNS FOR CLASSIFICATION (116 pp.)

Director of Thesis: Janusz A. Starzyk

The intent of this thesis is to improve on existing algorithms for determining classification rules by reducing the computational time to generate the reducts of an information system. Determining all reducts is an NP (Non-deterministic Polynomial time) complete problem and, therefore, as the data set grows in size, the time required for computation rapidly exceeds what is practical. This thesis has been able to significantly reduce the amount of time it takes to perform these computations. While the problem is still NP complete, the amount of time required by the methods introduced is less than other well-known methods provided by other software packages such as Rosetta [Ohr99] and RSES [RSES2].

Despite the reduct generation time improvements, larger databases still take far too long for effective reduct determination; therefore, heuristic non-exhaustive methods were also evaluated. In practical applications of rough sets, it is important that the obtained reducts retain most of the information about the original problem. In these applications, reducts of a dataset are used as classifiers to determine the “rules” for classification. The second half of this thesis proposes a method for rapidly producing effective classifiers of sufficient quality to get classification results of equal or better quality compared to exhaustive methods. The proposed method gives results that are at, or near, the same quality as those obtained from using the exhaustive method in only a fraction of the computational time.

Approved: _____

Janusz A. Starzyk

Professor of Electrical Engineering and Computer Science

Acknowledgements

I would like to acknowledge and thank all of the people that have assisted me with my Master's thesis at Ohio University. This thesis could not have been completed without their assistance.

First, I would like to thank my advisor Professor Janusz Starzyk for assisting me with my research. He has spent many hours advising me and has reviewed my work countless times. The members of my committee, Professor Savas Kaya, Professor Maarten Uijt de Haag, and Professor Xiaoping Shen, also deserve my thanks for taking the time out of their schedules to assist me in completing my Masters Degree.

I would also like to thank the Department of Electrical Engineering and Computer Science for their financial support in the form of a Teaching Assistantship.

Finally, I would like to thank my parents for their help. Their constant interest in my progress and willingness to read my work for clarity and correctness has been of great help.

Table of Contents

Page

Abstract 3

Acknowledgements 4

List of Tables 7

List of Figures 8

1. Introduction 9
   1.1 Background 9
   1.2 Research Objective and Thesis Organization 12

2. An Improved Exhaustive Reduct Determination Method 15
   2.1 Introduction 15
   2.2 Prior Research on Reducts 15
   2.3 Definitions and Notation 17
   2.4 The Reduct Generation Algorithm 19
   2.5 A Finding Reducts Example 23
   2.6 A Classification Based Example 28
   2.7 Improvements to the Reduct Generation Algorithm 36
   2.8 Testing of the Reduct Generation Algorithm 39
   2.9 Discussion of Results 41
   2.10 Conclusions 45

3. Generating Rules from Classifiers 47
   3.1 Introduction 47
   3.2 Rule Generation 47
   3.3 The Rule Generation Algorithm 49
   3.4 Examples 50

3.4.1 Random Example 51
3.4.2 The Iris Example 56
3.5 Closing Remarks 60
4. A Statistical Approach to Discerns in Classification 61
4.1 Introduction 61
4.2 Prior research on partial reducts 62
4.2.1 The Genetic Algorithm 62
4.2.2 Set Covering Heuristics 63
4.2.3 Approximate Hitting Sets 64
4.2.4 Dynamic Reducts 64
4.2.5 Ensemble Systems 65
4.2.6 Other Methods and Research 66
4.3 How Reducts Are Found Statistically 67
4.3.1 Population Sampling 70
4.3.2 Reducts and Discerns in the Sampling Method 72
4.4 The Hybrid Statistical Reduct Determination Algorithm 77
4.5 Results and Discussion 84
4.5.1 Discussion of Classification Results 88
4.5.2 Computational Time 92
4.5.3 Confidence Level and Confidence Interval Values 94
4.6 Comparison of Standard and Hybrid Statistical Methods 97
4.7 Conclusions 100
5. Thesis Conclusions 102
5.1 Conclusions 102
5.2 Future Work 104
References 106
Appendix – Full Tables 113

List of Tables

Page

Table 2.1 – Raw pseudo-random data. 29
Table 2.2 – Labeled data. 30
Table 2.3 – Duplicates removed and inconsistencies reclassified. 31
Table 2.4 – Discernibility list. 33
Table 2.5 – Reduced discernibility list – finding the core. 34
Table 2.6 – After removing rows containing the core feature(s). 35
Table 2.7 – Test with 150 instances, 30 features, and 2 classes. 38
Table 2.8 – Tested using RSES 2.2 algorithm. 40
Table 2.9 – Tested using improved exhaustive algorithm without prioritization 40
Table 2.10 – Tested using improved exhaustive algorithm with prioritization enabled 41
Table 2.11 – Reduct generation test for larger data sets. 44
Table 3.1 – Sorted data for columns 1 & 3. 51
Table 3.2 –Set of discovered rules for 1& 3. 52
Table 3.3 – Set of all possible reduct based rules. 53
Table 3.4 – Reduced set of all possible reduct based rules. 54
Table 3.5 – Rule based vote count. 56
Table 3.6 – Iris database rules. 57
Table 3.7 – Coverage values vs. correct classification of Iris database 58
Table 3.8 – Train/Test ratio vs. correct classification of Iris database. 59
Table 4.1 – Sampling for reducts. 75
Table 4.2 – Test results for ***wdbc*** database. 86
Table 4.3 – Test results for ***bcwis*** database. 86
Table 4.4 – Test results for ***wpbc*** database. 87
Table 4.5 – Test results for ***mushroom*** database. 87
Table 4.6 – Classification results for varying confidence values. 95
Table 4.7 – Standard method test results for ***mushroom*** database. 98
Table A1 – Full test results for the ***wdbc*** database. 113
Table A2 – Full test results for the ***bcwis*** database. 114
Table A3 – Full test results for the ***wpbc*** database 115
Table A4 – Full test results for the ***mushroom*** database 116

List of Figures

Page

Figure 2.1 – Reduct algorithm flowchart diagram.....22
Figure 2.2 – Computation time results using RSES. 42
Figure 2.3 – Computation time results using the improved exhaustive algorithm. 43
Figure 2.4 – Direct comparison of RSES results vs. the improved algorithm results. 43
Figure 4.1 – Column count difference vs. feature selection iteration. 76
Figure 4.2 – Flowchart of changes to the reduct algorithm. 80
Figure 4.3 – Exhaustive vs. statistical time to completion for ***wdbc*** database. 92
Figure 4.4 – Exhaustive vs. statistical time to completion for ***mushroom*** database. 93
Figure 4.5 – Average number of discerns vs. confidence interval. 96
Figure 4.6 – Discern generation time for the standard method. 99

1. Introduction

1.1 Background

In today's world, large volumes of data are routinely generated for various reasons. Collecting, correlating, and analyzing this data can be a tedious, time consuming, and complex job. Several methods have been created to streamline the process. Among them, rough set theory was developed and used in information systems and data mining to extract features, learn, classify and otherwise characterize the data by reducing the information necessary for its representation. Rough sets are a parallel concept to better-known fuzzy sets and are considered to be a better fit for algebraic representations and computer based discrete analysis. This thesis is focused on the methodology dealing with rough sets and more specifically, reducts.

During the early 1980s, Professor Zdzislaw Pawlak, the man who is considered to be the father of Rough Sets, pioneered the rough set concept of set approximation by using binary relations, based on the concepts of discernibility (ability to distinguish between objects). By using a rough set approximation, it is possible to build representative models of data. In simplistic terms, rough sets allow the placement of approximate boundaries where the actual location of the boundary based on known data is unknown. Consider a

solid circle in a painting program, for example. The program cannot display an exact circle, and when zoomed in enough individual pixels will be noticeable. The exact boundary of the printed image of a circle cannot be defined, but it can be approximated by the rough pixilation. Thus, a rough set has a rough boundary. There will be elements of a data set that are definitely members of the rough set and those that are definitely not. These are elements on the interior and exterior of the rough set. Their membership function is crisp (0 or 1) and known. Elements of the rough set boundary have an unknown membership function value. This is in contrast to fuzzy sets where membership values are known, but are anywhere from 0 to 1 (fuzzy). [Paw82]

In Rough Set Theory, a reduct is a subset of features of the greater dataset that, while reduced in size, contains the same information about the data as the full feature set and can still be used to differentiate every element within the entire data set. Reducts are useful because each reduct from a feature set provides a completely discernable picture of the data set. In particular, within the reduct defined subset there are no inconsistencies between the various data elements despite the reduction in the number of features present. Multiple reducts can be obtained for a given data set and the process of reduct generation is NP hard.

A good source for basic information about the underlying rough set theory and reducts is in Part II of the work by Aleksander Ohrn [Ohr99]. In his thesis he provides explanations of the benefits of discernibility based methods for analyzing medical data, and covers

much of the basics of reducts. As the creator of the Rosetta software, he needed to treat the work in depth. The work by Dale Nelson [Nel01] provides a good explanation of rough set theory and the generation of reducts while also applying the theory to Advanced Target Recognition (ATR). Herbert and Yao [Her05] show how reducts and other tools can be used for time-series analysis of stock exchanges. The paper is focused on finding complete sets of reducts as they provide better coverage of the rules that govern data dependencies and give statistical protection from noise included in the data set. Bjorvand and Komorowski [Bjo99] describe the use of genetic algorithms as a method of producing incomplete sets of reducts, and may be useful as a starting point for those who wish for less time consuming methods or to use larger data sets. A more thorough examination of reduct generation methods is presented in Section 4.2 of this thesis.

The majority of past and present work on the generation of reducts focuses on producing subsets of the set of all reducts in order to produce results in a manageable time frame. It was decided to go back and apply existing knowledge to the original problem by generating a complete set of reducts and then see what could be done to make the time expenditure more palatable. One may wonder why it would be worthwhile to generate an exhaustive set of reducts; one reason is simply for greater coverage of a set. When generating reducts the goal is often to use them to help generate rules to classify a significantly larger set. While it is true that a reduct can discern all the elements of the set it is related to, if one extracts the rules from that particular reduct the rules may not give

the expected coverage to the full data set. By including a greater number of reducts, or in this case, all of them, more rules relating to the set can be generated allowing for a subsequently better and more robust classification.

1.2 Research Objective and Thesis Organization

Reducts are commonly used to describe a larger dataset using a subset of the feature set because reducts contain the same information about the data as the full feature set and can still be used to differentiate every element within the entire set. The objective of this thesis is to improve on existing algorithms for determining reducts by reducing the computational time to generate all the reducts for an information system. The second objective is to improve upon existing reduct based classification methods. Existing algorithms are too slow, too computationally intensive, or not accurate enough. By trying to improve both exhaustive and non-exhaustive reduct generation methods, it is hoped that significant improvements can be made in both the time needed for reduct generation and in the accuracy in non-exhaustive reduct generation.

This thesis is organized into four chapters. Chapter 1 provides is the introduction and provides the reasoning behind this work as well as an outline of the following chapters. The bulk of this work is divided into two chapters (Chapters 2 and 4) whose overall goal, in both cases, is to present a simple, yet effective, method for producing good classifiers for use on databases in a reasonable amount of time. Chapter 2 provides additional

background information and some definition and notation relevant to reducts. Chapter 2 also describes an efficient way to determine complete sets of reducts. In this chapter, it is shown that the initial algorithm presented is successfully able to reduce the amount of time it takes to perform these computations. Thus, while the problem remains NP complete, the amount of time required by the developed method is less than the time required by exhaustive methods provided by others such as the well known Rosetta [Ohr99] and RSES [RSES2] software. Despite the time improvements, larger databases still take far too long for effective reduct determination due to the NP complete nature of the problem.

Unfortunately, simply generating reducts is not very useful; something needs to be done with the reducts to apply them in data mining problems. Chapter 3 describes methods for producing classification rules using reducts. It describes in detail the procedure used to generate the classification results that are presented in the subsequent chapter.

Chapter 4 uses the basic improvements shown in Chapter 2 and expands upon them in order to produce an even more effective method for the production of classifiers. It presents a simple, yet effective, method for producing good classifiers in a reasonable amount of time using the classification method covered in Chapter 3. Chapter 4 searches for a faster, but still simple method of determining reducts and was approached with the knowledge that any non-exhaustive method would be able to find only a fraction of the possible reducts in a reasonable amount of time. The fourth chapter presents a thorough

examination of a non-exhaustive method that uses a heuristic statistical approach to finding classifiers for use on information systems.

In practical applications of rough sets, it is important that the obtained reducts retain most of the information about the original problem. In Chapter 4, reducts of a dataset were used as classifiers to determine the “rules” for classification. Thus, not only was it necessary to find reducts faster, but a check of the results was required to demonstrate that they retain the same quality as a full set of reducts. Meaning, it was important to verify that this new method produced results at or near the same quality as the exhaustive method could. A check of the results from the proposed method shows that it maintains the quality of the exhaustive methods and yields computational times that are considerably shorter, even for vary large databases.

Chapter 5 briefly summarizes the major results and conclusions of this paper. It also suggests ideas to further improve the quality of the results as well as the computational time.

2. An Improved Exhaustive Reduct Determination Method

2.1 Introduction

In this chapter, the prior research on exhaustive reduct generation, and basic definitions and notation used in developing the new reduct generation procedure are discussed. This is followed by an explanation of the reduct generation algorithm, illustrated with a basic example and followed by a more complex classification based example. Subsequently, modifications to the basic algorithm and their effects on the algorithm's performance are discussed. Statistical testing compares this algorithm's performance with RSES using varying database sizes. The final sections of this Chapter present results and conclusions.

2.2 Prior Research on Reducts

By the nature of the problem, there are very few ways to implement an exhaustive reduct generation algorithm. Furthermore, because of the time needed for most exhaustive reduct generation problems to compute, the bulk (and all recent) research on reducts has been in regards to partial reducts (see Section 4.2). However, there remains a great deal

of research regarding the background of reduct generation and the more closely related alternative classification methods.

As previously mentioned Zdzislaw Pawlak pioneered Rough Set theory and introduced the concept of reducts [Paw82, Paw84, Paw85]. Furthermore, he provided the basic terminology and methodology for generating classification results from rough sets via reducts. Later, Skowron [Sko92] introduced the discernibility matrix as a means of representing an information system. The discernibility matrix is an effective, if somewhat memory intensive, tool used in finding reducts. Between Pawlak and Skowron most of the work regarding exhaustive reducts has been covered, and most other work has been on less thorough reduct generation methods.

However, there remain a few other methods somewhat related to reducts that can be examined, for example, fuzzy sets. Fuzzy sets, introduced by Zadeh in 1965 [Zad65] have an even longer history than rough set theory. Consider a grayscale image; the main difference between rough sets and fuzzy sets is that fuzzy sets focus more on the grayness of individual pixel, while rough sets focus more on the boundaries. To put it another way, a fuzzy set problem has a fuzzy imprecisely defined set membership function, while a rough set problem has imprecise or rough boundaries, hence the rough set. There has even been some work done combining the two [Dub90]. Other research on reducts pertaining to the non-exhaustive methods for reduct generation is referenced in Section 4.2.

2.3 Definitions and Notation

The following is an overview of the definitions, notation, and vocabulary for the reduct methods discussed in this thesis, and relies heavily on the notation as presented in some of Starzyk's earlier work [Sta99, Sta00].

Consider the information system (U, A, D), where $U = \{ x_1, ..., x_n \}$ is a nonempty finite set called the universe, and $A = \{a_1, ..., a_m\}$ is a nonempty set within U, *while* $D = \{d_1, ..., d_k\}$ *is the set of possible decisions that exist for U*. The elements of A, which are called attributes or features, are functions

$$a_i : U \rightarrow V_i, \tag{1}$$

where V_i is the value set of a_i. In a practical rough set system V_i is a discrete and finite set of values. For this work, a positive integer label over a user-specified range is used, usually something like V_i *={0, 1, …, 9}*. Elements, x_j, of U are called signals (or objects), thus a_i *assigns the value* $a_i(x_j)$ *to the signal* x_j.

The discernibility matrix, D_{dis}, of A is the $n \times n$ symmetrical matrix, which contains sets, c_{ij}, of attributes that differentiate signal x_i from x_j ($i \neq j$). D_{dis} is determined by comparing all "signals" with all other signals $i \neq j$. The symbol λ denotes that the particular comparison need not be considered because x_i and x_j ($i \neq j$) are of the same class.

$$D_{dis} = [c_{ij}], \text{ where } c_{ij} = \begin{cases} \{a \in A : a(x_i) \neq a(x_j)\} & if\ \exists\ d \in D[d(x_i) \neq d(x_j)] \\ \lambda & if\ \forall\ d \in D[d(x_i) = d(x_j)] \end{cases}. \quad (2)$$

Let $B \subseteq A$. The ***B-indiscernibility*** relation is

$$Ind(B) = \{(x,y) \in U \times U : (\forall a \in B)(a(x) = a(y))\} \quad (3)$$

Essential for the information system are the reducts that describe knowledge represented in this system. A set $B \subseteq A$ is a **discern** in A if $Ind(B) = Ind(A)$. A discern is called a **reduct** if $(\forall a \in B)\ Ind(B - \{a\}) \supset Ind(B)$, where "$\supset$" denotes a proper subset relation. The set of all reducts of A is denoted $Red(A)$. Thus, if a set F is a Discern then $\exists B \in \mathrm{Re}\,d(A)$ and $B \subset F$. In addition, if A is a reduct then for each a, $b \in A\ \ Dis(\{a\}) \neq Dis(\{b\})$.

In practical applications, a reduct is a subset of features of a dataset that despite being smaller than the full dataset preserves discernibility over the universe U of the full dataset. The core of the information system is defined as a set $P \subseteq A$ such that

$$P = \bigcap_{B \in \mathrm{Re}\,d(A)} B \quad (4)$$

and a set S is a **shell** if

$$\exists B \in \mathrm{Re}\,d(A) \quad P \subset S \subset B \quad (5)$$

Thus, a shell contains core attributes necessary to make a reduct, yet it does not remove all inconsistencies. A shell can be considered the intermediate stage on the path toward a reduct. Because of this, in this work a shell is sometimes referred to as the **scan path** or **search sequence**.

This chapter aims at developing a more efficient reduct generation procedure. The reduct generation procedure developed in [Sta99] is based on the expansion of the discernibility function into a disjunction of its prime implicants by applying the absorption or multiplication laws. This procedure is not sufficiently efficient to allow us to use it with real-world size problems. The approach taken here is to simplify reduct generation by first appending the core set with attributes that remove most of the inconsistencies left in the information system until all the inconsistencies are removed to obtain discerns B_m. This method makes use of a simple discernibility matrix based heuristic feature selection algorithm and several other minor and major additions that will be discussed further on.

$$B_i = B_{i-1} + a_i; \quad card\big(Ind(B_{i-1} + a_i)\big) = \min_k card\big(Ind(B_{i-1} + a_k)\big) \quad i = 0,...,m$$
$$where \quad B_0 = P \tag{6}$$

Once a discern is obtained by this method, it is verified as to whether or not it is actually a reduct. If not, it is simply not added to the set of reducts that have been found.

2.4 The Reduct Generation Algorithm

This section presents an efficient reduct generation algorithm based on discerns derived from the discernibility matrix. In the following algorithm, the term “search sequence” is used several times. The search sequence consists of the sorted set of features yet to be checked as possible reduct elements at the current search position. It is used in a recursive

manner to perform a complete search for reducts. In terms of the preceding notation, the search sequence is the current shell.

Since much of the computation occurring deals directly with the discernibility matrix, it was decided to modify it into a more computationally viable form. This form is called the discernibility list or D_{list}. The discernibility list performs the same function as the discernibility matrix, but "flattens" the discernibility matrix into a single column of sets. Furthermore, to ease computation, the sets represented by c_{ij} are transformed into Boolean rows. For example, a set $\{a_1, a_3\}$ becomes $\{1 \quad 0 \quad 1 \quad 0\}$ when there are 4 features. This has two advantages; first, it simplifies the process of scanning and reducing the discernibility data and second, it reduces the data structure size. (A discernibility matrix is a square matrix, but there is no use for the diagonal or the duplicated results that exist with $c_{ij} = c_{ji}$. See the example following this section for an example of the discernibility list vs. a discernibility matrix.)

The basic steps in the reduct generation algorithm are as follows:

1. Ensure the data set is scaled to the desired range.
2. Remove duplicates and reclassify initial inconsistencies.
3. Generate the discernibility list and find the core feature(s).
4. Remove rows containing the core feature(s) from the discernibility list.
5. Determine reducts by recursively repeating steps 6-12:
6. Reorder search sequence by feature representation in remaining discernibility list.
7. Repeat steps 8-12, *n* times with *k* being the current iteration, where *n* is the size of the remaining search sequence. The search sequence is: all the features – Core –

existing path (features already scanned), and *k* is used to the current prioritized feature to be examined.

8. Remove rows containing feature *k* from the discernibility list. If no rows are removed, return to step 7 and check the next *k*.
9. If the discernibility list is empty, i.e. there are no inconsistencies left, a Discern has been found and it may be a Reduct. Continue to step 10 to verify the reduct, otherwise go to step 11.
10. If removing any of the earlier features in the present search sequence does not introduce inconsistencies, then it is not a Reduct. Go to step 12.
11. Since inconsistencies are still present, recursively call from step 5. Pass to Step 5 the new search sequence and set of features to be examined minus the feature *k* just added (Step 7). Continue to step 12 when the recursive call completes.
12. If the Discern was a valid Reduct add it to a Reduct set to be passed back up the recursive call path. If $k < n$ increment k and go to step 8 as was specified in step 7, otherwise, continue to step 13.
13. Move back up the recursive path. (Return to step 12.)

Using this algorithm provides a significant improvement in time to completion over the computation provided by RSES [RSES2] and other exhaustive reduct determination algorithms. The primary reason for this improvement is the use of a discernibility list and the prioritization of the scanning of features. Figure 2.1 below gives a visual representation of the algorithm.

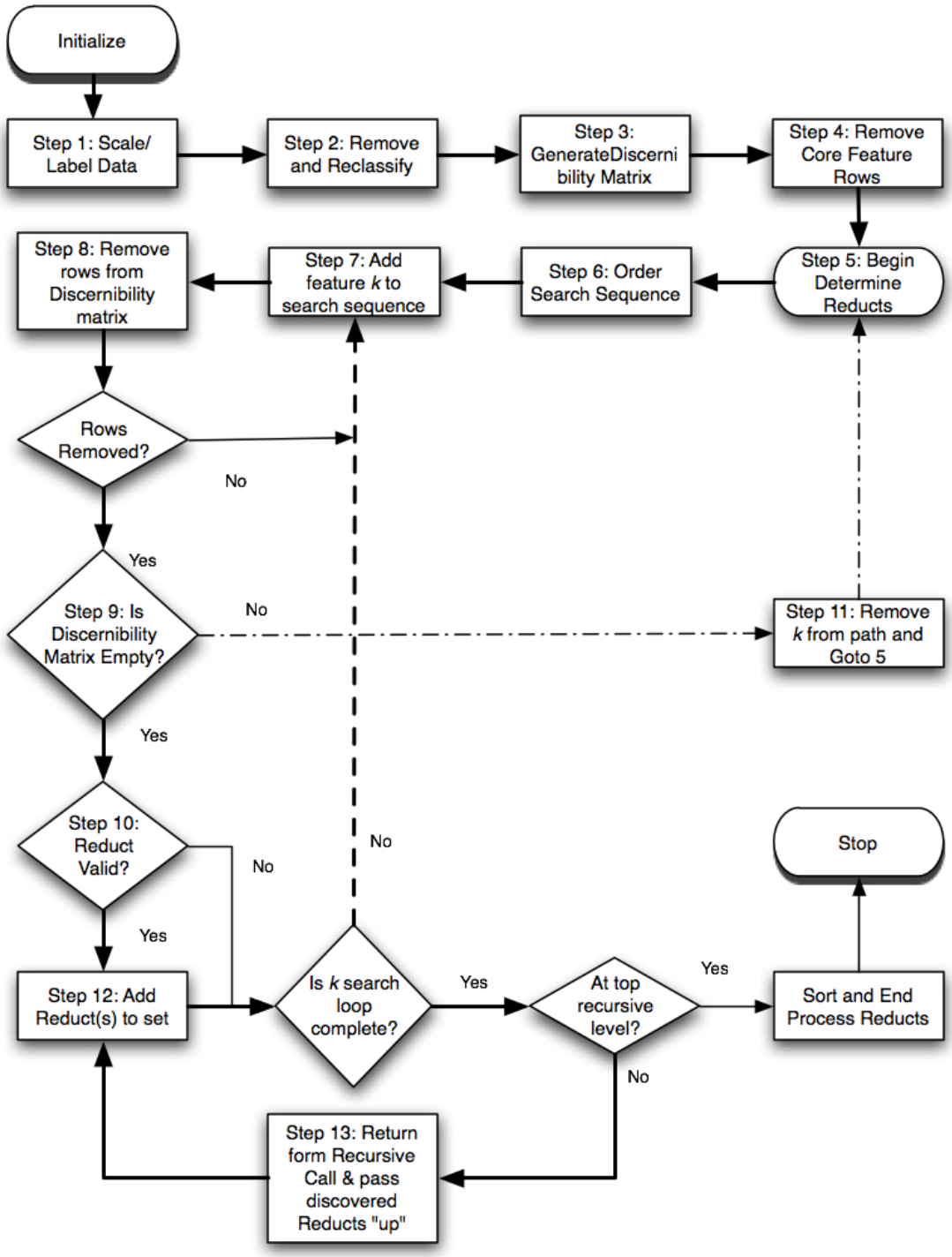

Figure 2.1 – Reduct algorithm flowchart diagram.

2.5 A Finding Reducts Example

To illustrate this approach a simple example is used. Consider the Decision System (U, A, T) where

$$U = \{x_1, x_2, x_3, x_4\}$$
$$A = \{a_1, a_2, a_3, a_4\}$$

$$T = \begin{pmatrix} dog & white & 3 & s \\ cat & white & 3 & s \\ horse & yellow & 0 & s \\ cat & red & 0 & s \\ dog & red & 3 & t \end{pmatrix}$$

U is the universe, A is the set of attributes, T is the decision table (A decision table contains values $v_{ij} = a_j(x_i)$ as defined in (2)), and D_{dis} is the discernibility matrix. Discernibility matrix, D_{dis}, differentiates between all signal pairs in T as follows:

$$D_{dis} = \begin{bmatrix} \{a_1\} & \{a_1, a_2, a_3\} & \{a_1, a_2, a_3\} & \{a_2, a_4\} \\ & \{a_1, a_2, a_3\} & \{a_2, a_3\} & \{a_1, a_2, a_4\} \\ & & \{a_1, a_2\} & \{a_1, a_2, a_3, a_4\} \\ & & & \{a_1, a_3, a_4\} \end{bmatrix}$$

Next, let us take the discernibility matrix and transform it into a discernibility list by removing the redundant empty spaces, flattening the sets into a single column, and transforming the sets into Boolean representations.

$$D_{list} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

There are two minimal reducts to be found in this system.

$$Reduct_1 = \{a_1, a_3, a_4\}$$
$$Reduct_2 = \{a_1, a_2\}$$

The discernibility list is used to generate Discerns. Each time a new attribute/feature (with a non-zero column in the discernibility list) is selected it resolves some inconsistencies. This can be represented by removing those rows from the discernibility list that have a '1' in the columns that correspond to the selected attributes.

A walk-through of the procedure using the steps of the reduct generation algorithm (refer to Section 2.3):

Step 1: Ensure the data is properly scaled. Each attribute should have a small number of discrete values. If the attributes are related to integer or real values, they must be first subdivided into a few disjoint literals.

Table T is already scaled well enough as it does not contain decimal numbers or a hundred different values.

Step 2: Remove duplicates and reclassify initial inconsistencies.

There are no duplicates or inconsistencies in this data set.

Step 3: Generate the discernibility list and find the core feature(s).

D_{list} is the discernibility list. The $Core=a_1$ since row 1 contains only 1 element and the element is in column 1, thus making column 1 part of the core.

Step 4: Remove rows containing the core feature(s) from the discernibility list.

If all rows containing a '1' in column 1 are removed, the result is

$$D_{list} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Mathematically one can describe this by saying $|D_{list},X|$ = *the number of rows/inconsistencies remaining* in the discernibility list after the rows associated with the selected set of attributes, X, are removed.

$|D_{list},Core| = 2 \neq 0$ so the remaining set of attributes is considered $\{a_2, a_3, a_4\}$.

Step 5: Begin recursive reduct determination by repeating steps 6-12:

Step 6: Reorder search sequence by feature representation in remaining discernibility list.

This can be done by simple summing the columns, ending with a 2 for columns 2 and 1's for the remaining columns. Since this is the existing search sequence order, nothing will be changed.

Step 7: For k = 1 to n, where n is the size of the remaining search sequence. The search sequence is: all the features – Core – existing path (features already scanned), and k points to the current prioritized feature to be examined. In this case, the search sequence is: $\{a_1, a_2, a_3, a_4\} - \{a_1\} - \{\varnothing\} = \{a_2, a_3, a_4\}$.

So k=1 to 3. (with k=1 pointing to a_2, k=2 pointing to a_3, and so on.)

Essentially, $Path = P = Core = \{a_1\}$ and set $r = \{a_2\}$ have been set, where r is the candidate element for the reduct.

Step 8: Remove rows containing feature k from the discernibility list. If no rows are removed, return to step 7 and check next k.

Since a_2 has a '1' in both rows, both rows are removed from the D_{list} matrix.

Then,

$$\left|D_{list}, \{r\}\right| = 2 \Rightarrow \min\left(\left|D_{list}, P\right|, \left|D_{list}, \{r\}\right|\right) = 2$$

and

$$\left|D_{list}, P \cup \{r\}\right| = 0.$$

Step 9: If the discernibility list is empty, i.e. there are no inconsistencies left, a Discern has been found and may be a Reduct.

Step 10: Verify Reduct – If removing any of the earlier features in the present search sequence does not introduce inconsistencies then it is not a Reduct.

This step takes the matrix from step 4 and removes features earlier in the path to see if inconsistencies are, or are not, introduced. In this case, a_2 is the only non-core element in the path and its removal will introduce inconsistencies, so it is verified that $\{a_1,a_2\}$ is a reduct. If the Discern is a valid Reduct then go to Step 12. If the Discern is not a valid Reduct (inconsistencies are not introduced by removing at least one of the features from the Discern), then go to Step 11.

Step 12: If the Discern was a valid Reduct, add it to a Reduct set to be passed back up the recursive call path.

The newly found reduct is added to a set to later be returned up the recursive path.

Step 13: Move back up the recursive path.

This would be done if the **For** loop (steps 7-12) were complete, however, k=2 and k=3 still need to be evaluated. So the calculation moves back to step 8 with k=2

Step 8: (Now evaluating a_3) Since a_3 has a '1' present in the second row the D_{list} matrix that row is removed. The search sequence is currently: $\{a_1, a_2, a_3, a_4\} - \{a_1\} - \{a_2\} = \{a_3, a_4\}$.

$$D_{list} = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}$$

Step 9: The discernibility list is not empty so go to step 11.

Step 11: Else – Inconsistencies are still present.

Recursively call from step 5. Pass the new search sequence and set of features to be examined minus the one just added to the path.

The search sequence to be passed will be $\{a_3\}$, and the set of features to be examined will contain only $\{a_4\}$

Step 5: Begin a new recursive level.

Step 6: Reorder the search sequence. Nothing needs to be done here since the path currently contains only a_4.

Step 7: For k=1 to 1, with k=1 pointing to a_4. The search sequence is currently: $\{a_1, a_3, a_4\} - \{a_1\} - \{a_3\} = \{a_4\}$. The attribute a_2 is missing because it is not part of the search sequence in the upper level and so is not passed to subsequent recursions.

Step 8: Since a_4 has a '1' present in the only remaining row the D_{list} matrix that row is removed leaving the matrix empty

Step 9: The matrix is empty, therefore go to step 10.

Step 10: Now the reduct is verified $\{a_1, a_3, a_4\}$ by removing features to see if they introduce inconsistencies. (The core a_1 does not need to be verified because it is already known that removing it will introduce inconsistencies.) In this case removing a_3 will introduce inconsistencies so this reduct can be considered verified. (a_4 does not need to be checked because it was the last feature added and it is already known that removing it will introduce inconsistencies.)

Step 12: The discern was valid so it is added to a reduct set.

Step 13: The For-loop is complete, so the result is passed back up the recursion path.

Step 12: The reduct from the recursion is added to the existing set.

Step 13: Finish the For-loop and examine k=3.

Step 8: (Now evaluating a_4). Since a_4 has a '1' present in the second row the D_{list} matrix that row is removed. The search sequence is currently: $\{a_1, a_2, a_3, a_4\} - \{a_1\} - \{a_2, a_3\} = \{a_4\}$.

$$D_{list} = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$$

Step 9: The discernibility list is not empty so go to step 11.

Step 11. Else – Inconsistencies are still present.

Recursively call from step 5. Pass the new search sequence and set of features to be examined minus the one just added to the path.

However, the set of features to be examined is empty since a_4 is now being observed, and there are no other features in the search sequence.

And as $\{a_1,a_4\}$ cannot be a reduct, the step halts at this point. And go to step 13.

Step 13: Move back up the recursive path.

At this point the reduct set contains $\{a_1,a_2\}$ & $\{a_1,a_3,a_4\}$. Because the recursion is at the top level, these reducts represent the full set and are passed back to the main program.

The preceding example illustrates the reduct finding process, but lacks one significant aspect, classes. In most real-world problems there is some element of classification applied to the data set. The next, more complex, example illustrates how the algorithm is actually implemented and how classifications are taken into account.

2.6 A Classification Based Example

In the following example, the creation of a discernibility list is described in detail and the theory described above is applied to illustrate a reduct determination problem. This example is larger, contains labeling and data classifications, and is more in line with what would be expected in a real-world problem. In real-world problems, feature values often come from continuous functions rather than discrete symbolic categories. Such data may describe a typical set of samples with multiple attributes and the problem is to classify these samples based on the observed attribute vales.

The first step in determining the reducts of such a dataset is scaling, or as it is sometimes called, labeling of the data. Data with symbolic values can be labeled directly, although assigning different numerical labels to symbols results in different data distributions in multidimensional feature spaces and may affect the complexity of the classification process. Floating point data needs to be scaled to a specific range to be properly categorized. Typically, a nonlinear scaling is applied to render the scaled data to a uniform distribution to facilitate classification. Table 2.1 represents a set of pseudo-random data. The data for each column can be of any range or type of value, however, for demonstration's sake, this simple set of data was created.

Table 2.1 – Raw pseudo-random data.

Sample	Feature 1	Feature 2	Feature 3	Feature 4	Classification
1	.123	.625	.901	.321	1
2	.747	.085	.897	.328	1
3	.101	.605	.925	.293	1
4	.585	.256	.511	.876	2
5	.692	.707	.121	.192	2
6	.321	.606	.901	.307	1
7	.295	.600	.935	.295	2
8	.567	.211	.898	.487	1
9	.075	.815	.456	.815	1
10	.469	.406	.488	.907	2

The labeling process may be based on a simple linear scale using generally recognized units – for instance shoe size, or may be a result of a more complex labeling process - for

instance based on data entropy. Table 2.2 displays the data after it has been linearly scaled and labeled in a range from 1-5. That is, the range 0 to 0.199 is given the scaled value of 1; the range from 0.2 to 0.399 is given the scaled value of 2 and so forth. This is not necessarily the best labeling method available, but was used for simplicity's sake.

Table 2.2 – Labeled data.

Sample	Feature 1	Feature 2	Feature 3	Feature 4	Classification
1	1	4	5	2	1
2	4	1	5	2	1
3	1	4	5	2	1
4	3	2	3	5	2
5	4	4	1	1	2
6	2	4	5	2	1
7	2	4	5	2	2
8	3	2	5	3	1
9	1	5	3	5	1
10	3	3	3	5	2

Once the dataset has been scaled, redundant information is removed to reduce unnecessary computation time. Specifically, duplicate entries are removed, whether they were caused by scaling or were present in the original data. Furthermore, so that there are no inconsistencies in the newly scaled data, identical entries with different classes can either be removed or reclassified to a new, different class. Removing them is simpler, but at the cost of reducing the resolution of the results. By reclassifying the inconsistencies to a new class, their impact on the dataset is retained. In Table 2.3, rows 3 and 7 of Table 2.2 are missing. Row 3 was identical to row 1 and was removed. Rows 6 and 7 possess

identical features, but different classifications. (This would tend to happen near "boundaries" within a dataset where values are close, but classifications are different.) Therefore, row 7 was removed, and row 6 was reclassified to Class=3.

Table 2.3 – Duplicates removed and inconsistencies reclassified.

Sample	Feature 1	Feature 2	Feature 3	Feature 4	Classification
1	1	4	5	2	1
2	4	1	5	2	1
4	3	2	3	5	2
5	4	4	1	1	2
6	2	4	5	2	3
8	3	2	5	3	1
9	1	5	3	5	1
10	3	3	3	5	2

The next step is to find the core of the data set. The core consists of the features that are common to all reducts. Core features are features that cannot be removed from the data set without introducing new inconsistencies. There are two principal ways to determine the core. The most straightforward method (in that it is the first method most people would initially attempt) is to simply remove one feature at a time and check for inconsistencies. If any inconsistencies show up, the feature is a core feature and must not be removed to maintain the initial feature set discernibility property. The other method is to build a discernibility list and determine the core from that matrix by looking for rows in the discernibility list that contain only 1 element, indicating that this specific element is the only thing differentiating the two signals represented by the discernibility list. The

straightforward method, while obvious, is time consuming as a result of the need to constantly check for inconsistencies. In contrast, building a discernibility list places all possible inconsistencies in one location and does not need to be regenerated time and again.

Because of the way the discernibility list is built, the fact that the maximum number of rows is $n(n-1)/2$ is known, or is in this instance, 45. This maximum size matrix results from comparisons between all signals of the data set. For example, in a 10 signal data set, signal 1 is compared with signals 2 through 10, signal 2 is compared with signals 3-10, and so on. The comparison consists of comparing the feature elements of each signal pair to see if they are equal or not. Since the point of interest is in differences between signals, different features are indicated by a '1', while equivalent values are indicated by a '0'. This allows a Boolean discernibility list to be built. Using a Boolean matrix saves memory and allows for operations on the discernibility list to be performed faster than if the same or equivalent operations were done by another method. Table 2.4 shows the discernibility list generated based on data from Table 2.3.

Once the discernibility list has been created, it is a simple process to determine the core. First, in order to speed computations, both now and during future calculations, comparisons between signals of the same class need to be removed. These comparisons provide no useful information. With respect to determining the core, they are not relevant because the lack of a "core feature" in a discernibility list entry will not introduce any

inconsistencies in the data set. Nor, for the same reason, is it necessary to make comparisons between signals of the same class for finding reducts later on. The reduct determination process is not concerned with differences between signals of the same class, only between signals of different classes. Thus, all the following comparisons can be removed: 1-2, 1-8, 1-9, 2-8, 2-9, 4-5, 4-10, 5-10, and 8-9.

Table 2.4 – Discernibility list.

Comparison	Feature 1	Feature 2	Feature 3	Feature 4	Same Class?
1-2	1	1	0	0	1
1-4	1	1	1	1	0
1-5	1	0	1	1	0
1-6	1	0	0	0	0
1-8	1	1	0	1	1
1-9	0	1	1	1	1
1-10	1	1	1	1	0
2-4	1	1	1	1	0
2-5	0	1	1	1	0
2-6	1	1	0	0	0
2-8	1	1	0	1	1
2-9	1	1	1	1	1
2-10	1	1	1	1	0
4-5	1	1	1	1	1
4-6	1	1	1	1	0
4-8	0	0	1	1	0
4-9	1	1	0	0	0
4-10	0	1	0	0	1
5-6	1	0	1	1	0
5-8	1	1	1	1	0
5-9	1	1	1	1	0
5-10	1	1	1	1	1
6-8	1	1	0	1	0
6-9	1	1	1	1	0
6-10	1	1	1	1	0
8-9	1	1	1	1	1
8-10	0	1	1	1	0
9-10	1	1	0	0	0

The core is found by checking the resulting discernibility list (Table 2.5) for rows that contain only one difference. In a row with only one difference, the associated feature is a core feature. It is the only feature that differentiates the two signals of different classes that were compared to create the entry in the discernibility list. In Table 2.5, the only such case is with comparison 1-6, Feature 1. Therefore, Feature 1 is the core of the data set. It should be noted that the more classes are present in a data set, the longer it will take to process the data set. This is because more classes means there are fewer comparison between signals of the same class, thus allowing for a discernibility list closer to the maximum size specified by $n(n-1)/2$. It follows then, that the larger the discernibility list is, the longer a reduct search is likely to take.

Table 2.5 – Reduced discernibility list – finding the core.

Comparison	Feature 1	Feature 2	Feature 3	Feature 4
1-4	1	1	1	1
1-5	1	0	1	1
1-6	**1**	**0**	**0**	**0**
1-10	1	1	1	1
2-4	1	1	1	1
2-5	0	1	1	1
2-6	1	1	0	0
2-10	1	1	1	1
4-6	1	1	1	1
4-8	0	0	1	1
4-9	1	1	0	0
5-6	1	0	1	1
5-8	1	1	1	1
5-9	1	1	1	1
6-8	1	1	0	1
6-9	1	1	1	1
6-10	1	1	1	1
8-10	0	1	1	1
9-10	1	1	0	0

Next it is time to find the reducts. It is known that Feature 1 will be present in all of the reducts, so all rows of the current discernibility list that contain a difference related to Table 2.1 can be removed, for example rows 1-4 through 2-4. This can be safely done because each row of the matrix is an individual comparison; selecting *any* feature of that row will differentiate the two associated signals, thus ensuring, that there will be no inconsistencies associated with that particular comparison. Another way to look at the situation is to see the discernibility list as a list of all possible inconsistencies, which, by selecting features for reducts, are eliminated bit by bit.

Table 2.6 – After removing rows containing the core feature(s).

Comparison	Feature 1	Feature 2	Feature 3	Feature 4
2-5	0	1	1	1
4-8	0	0	1	1
8-10	0	1	1	1

Table 2.6 shows what remains of the discernibility list after all rows representing signal comparisons with differences at Feature 1 are removed. With only 3 rows remaining, the set of reducts can easily be determined. Based on Table 2.6, the set of reducts is [{1,3} {1,4}]. If Feature 2 is selected, then either Feature 3 or 4 can be selected to differentiate between all signals from different classes. However, they can also be differentiated by selecting just Feature 3 or 4, therefore, {1,2,3} and {1,2,4} are not reducts. Since both

selected reducts are minimal, there are 2 minimal reducts in this set of data. Any one of these reducts can represent the original dataset without any loss in classification.

2.7 Improvements to the Reduct Generation Algorithm

The introduction of the discernibility list simplified reduct generation compared to direct use of the decision table. This alone resulted in a significant speed up of calculations compared to classical reduct generation algorithms (e.g. Rosetta). Several modifications and improvements to this algorithm speed up the computations even more. The most significant of these improvements is prioritization of the scanning order.

Modification 1: Prioritization of the search sequence by representation in discernibility list

This modification was chosen for the simple reason that it was thought that better and shorter reducts would be generated sooner with less computational overhead. Results seem to have shown this assumption to be correct. Prioritization is accomplished at the beginning of the recursive scanning algorithm. After the discernibility list is modified and the algorithm is called, a representation of each feature in the discernibility list is calculated by summing the number of 1's in each column. The results are then sorted, so that the feature with the highest representation is checked first. This has the effect of not only choosing the feature that will most likely be present in the majority of reducts, but also reduces quickly the size of the discernibility list. It improves the results for less highly represented features, because many unsuccessful trails with these features are avoided.

Modification 2: Reduce recursion when it is not needed.

This modification eliminates recursion when the feature to be added to the search sequence is represented in all rows of the remaining discernibility list. Otherwise, the algorithm would reduce the discernibility list to an empty set, then recursively call itself only to determine what is already known, that because the feature fully covered the discernibility list it was a discern. This modification has minimal impact on the random matrixes tested – only very minor improvements were shown. However, it does save some time and memory by cutting down on unnecessary execution of code.

Modification 3: Early elimination of inconsistencies in the remaining part of discernibility list.

When scanning, the program checks to see if the column count (sum of all remaining 1's) of the discernibility list is greater than or equal to the number of rows of the discernibility list. If it is not, then it can no longer be possible to find any reducts with the remaining discernibility list. This check seems to take more time than it saves (at least for the random sets tested) and for all of the various data set sizes.

Modification 4: Finding empty rows in the discernibility list.

Just before the recursion call stage the program checks if there are any empty rows in the discernibility list. If so then the row cannot be removed because all features that would differentiate the pair of signals represented by it have already been examined and cannot be added to the reduct. Checking for empty rows saves time compared to going through further recursions to the point only empty rows remain. This is actually a more computationally intensive version of the Modification 3 that can catch what #3 may miss.

Modification 5: **Limit the scan depth when searching for a minimum reduct.**

This is useful if the objective is primarily to find the minimum size reduct(s) and a rough size of the selected database's reducts is known. Limiting the scan depth saves time because it eliminates larger reducts and longer path searches.

Table 2.7 illustrates the effects on computation time of these modifications to the basic algorithm. Tests were done on two computers. The first was a PowerMac G5 Dual 2.5GHz machine, while the second was a 2.0 GHz Dell Inspiron 6000. Rows 1-5 show the results when only a single modification is active per run. It would seem, based on these results, that modifications 3 & 4 have a detrimental effect when used by themselves, while modification 1 provides the greatest improvement and modification 2 provides a more modest improvement. Entries in rows 6-9 illustrate the cumulative effect of enabling one modification at a time until all are present in row 9. Interestingly, modifications 3 & 4 seem to improve the overall time when working in concert with the other modifications as seen in the declining times in rows 7 and 8.

Table 2.7 – Test with 150 instances, 30 features, and 2 classes.

Test	Code Modification Enabled	PowerMac G5 Time (secs)	Inspiron 6000 Time (secs)
1	All off	140.46	436.94
2	1	99.54	240.84
3	2	137.62	429.75
4	3	144.89	438.08
5	4	142.2	434.81
6	1&2	97.49	241.281
7	1, 2 & 3	88.98	235.03
8	1, 2, 3, & 4	86.52	234.35
9	1, 2, 3, 4 &5	80.51	230.50

Because the most significant effects were found with the first proposed modification, “Prioritization of scan order by representation in discernibility list”, subsequent efforts are focused on the improvements resulting from this modification only.

2.8 Testing of the Reduct Generation Algorithm

The following three tables illustrate the performance of the Reduct Generation Algorithm. All three tables use the same data set: a single data file of 500 rows and 40 columns with integer values of 0-8 for the feature elements. The data set was modified as needed; columns/rows were removed or classifications were changed, but the same file was used as the basis for all tests. The data was generated on a 2GHz Dell Inspiron 6000 with 1.25GB of RAM.

Tables 2.8-2.10, show the time to completion and the number of reducts found for different numbers of instances, classes, and attributes. Table 2.8 shows the time for calculating reducts using RSES 2.2. RSES was chosen as a standard because it is a well-known program designed for use with Rough Sets and is considered faster than its “competitor”, Rossetta. Tables 2.9 and 2.10 contain comparable results generated by the algorithm presented in this thesis work. (The algorithm had Modification 5 disabled so that all reducts would be generated to give an accurate comparison to RSES.) Notice that the results for RSES ran into the tens of thousands of seconds. The times were so large that Table 2.8 was not completed, because for larger numbers of rows and columns the

time to run the algorithm became too great to be useful. For instance, the test with 30 attributes, 75 instances, and 4 classes took 50737 seconds, or roughly 14 hours.

Table 2.8 – Tested using RSES 2.2 algorithm.

	Rows	50	75	100	125	150
Columns (Attributes)	#Classes	Time(s) (#Reducts)	Time(s) (#Reducts)	Time(s) (#Reducts)	Time(s) (#Reducts)	Time(s) (#Reducts)
20	2	19 (1017)	90 (2309)	233 (2680)	449 (2656)	769 (3193)
	3	39 (1406)	178 (2661)	451 (2651)	982 (4381)	1756 (4655)
	4	68 (1867)	307 (2630)	779 (2884)	1510 (4381)	2692 (6608)
25	2	323 (2861)	1912 (6543)	5497 (7186)	11184 (7456)	23350 (10282)
	3	546 (3892)	3574 (7002)	11566 (7353)	N/A	N/A
	4	651 (4357)	4110 (7307)	N/A	N/A	N/A
30	2	2528 (5723)	23785 (14023)	N/A	N/A	N/A
	3	5323 (8652)	37300 (15326)	N/A	N/A	N/A
	4	6428 (9851)	50737 (15544)	N/A	N/A	N/A

Table 2.9 – Tested using improved exhaustive algorithm without prioritization.

	Rows	50	75	100	125	150
Columns (Attributes)	#Classes	Time(s) (#Reducts)	Time(s) (#Reducts)	Time(s) (#Reducts)	Time(s) (#Reducts)	Time(s) (#Reducts)
20	2	1.156 (1017)	2.938 (2309)	6.141 (2680)	11.297 (2656)	19.813 (3193)
	3	1.469 (1406)	4.391 (2661)	9.391 (2651)	18.484 (3175)	33.563 (4655)
	4	2.141 (1867)	6.625 (2630)	14.906 (2884)	30.625 (4381)	54.531 (6608)
25	2	2.781 (2861)	8.000 (6543)	18.078 (7186)	38.500 (7456)	72.469 (10282)
	3	3.469 (3892)	12.375 (7002)	29.547 (7353)	62.016 (9753)	118.594 (15193)
	4	3.875 (4357)	14.297 (7307)	34.875 (7484)	77.453 (11279)	151.063 (18095)
30	2	5.453 (5723)	20.688 (14023)	50.266 (15432)	113.016 (18739)	222.594 (29812)
	3	7.766 (8652)	30.484 (15326)	77.875 (16524)	174.438 (23870)	341.984 (32767)
	4	9.484 (9851)	38.766 (15544)	100.484 (17487)	227.688 (29233)	450.313 (49004)

Table 2.10 – Tested using improved exhaustive algorithm with prioritization enabled.

	Rows	50	75	100	125	150
Columns (Attributes)	#Classes	Time(s) (#Reducts)	Time(s) (#Reducts)	Time(s) (#Reducts)	Time(s) (#Reducts)	Time(s) (#Reducts)
20	2	0.688 (1017)	1.844 (2309)	3.563 (2680)	6.156 (2656)	11.828 (3193)
	3	0.984 (1406)	2.609 (2661)	5.063 (2651)	10.516 (3175)	21.031 (4655)
	4	1.359 (1867)	3.594 (2630)	8.156 (2884)	18.641 (4381)	36.563 (6608)
25	2	1.594 (2861)	4.625 (6543)	9.188 (7186)	18.172 (7456)	38.188 (10282)
	3	2.234 (3892)	6.547 (7002)	14.250 (7353)	32.531 (9753)	69.484 (15193)
	4	2.516 (4357)	7.422 (7307)	16.781 (7484)	42.891 (11279)	89.797 (18095)
30	2	2.938 (5723)	9.906 (14023)	21.063 (15432)	48.906 (18739)	109.703 (29812)
	3	4.500 (8652)	13.984 (15326)	32.672 (16524)	83.188 (23870)	183.047 (32767)
	4	5.314 (9851)	16.750 (15544)	41.234 (17487)	109.906 (29233)	241.094 (49004)

2.9 Discussion of Results

Comparing the RSES results of Table 2.8 to Table 2.9, at low attribute levels, a reduction of over 25x in computation time has been achieved. With 30 attributes and 75 instances, a computation time reduction of about 1:2200 was obtained. Comparing the entries of Table 2.9 to those of Table 2.10, shows that prioritization further reduced computation time by more than a factor of two. The number of reducts is shown to illustrate that the shift in computational method did not lead to errors in the numbers of reducts found.

Figures 2.2-2.4 graphically illustrate the improvement of this algorithm vs. RSES. Figure 2.2 shows results generated using two classes and two different numbers of attributes. Figure 2.3 similarly shows generated using two classes and four different numbers of

attributes. The purpose of Figure 2.4 is to visually compare the first set of RSES results at 20 attributes and 2 classes with the improved algorithms results at 20 & 25 attributes with 2 classes.

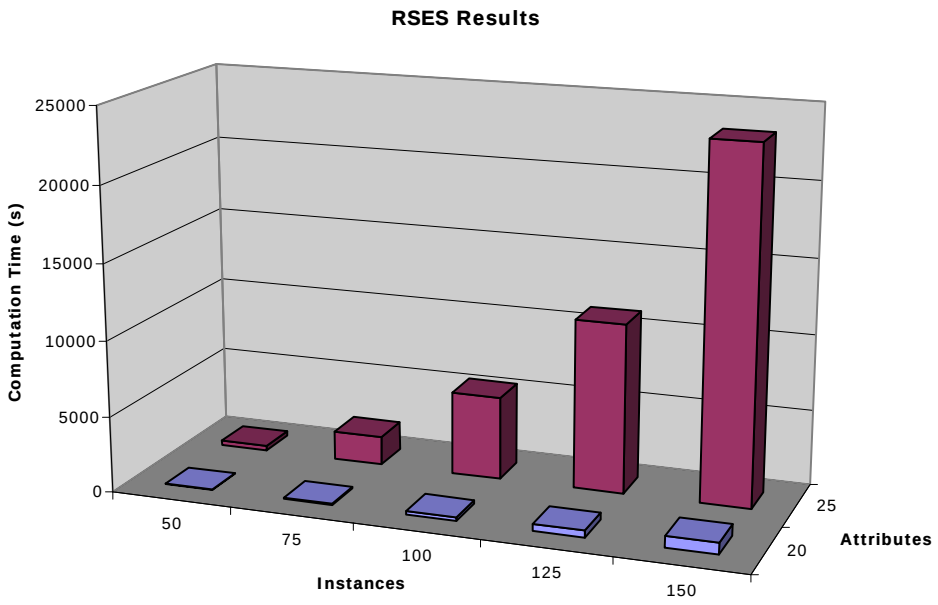

Figure 2.2 – Computation time results using RSES.

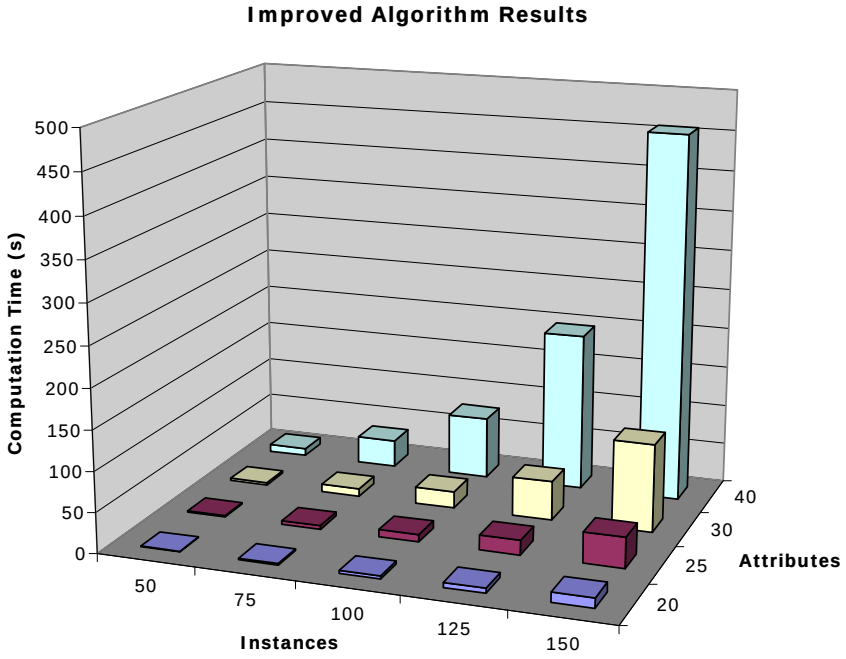

Figure 2.3 – Computation time results using the improved exhaustive algorithm.

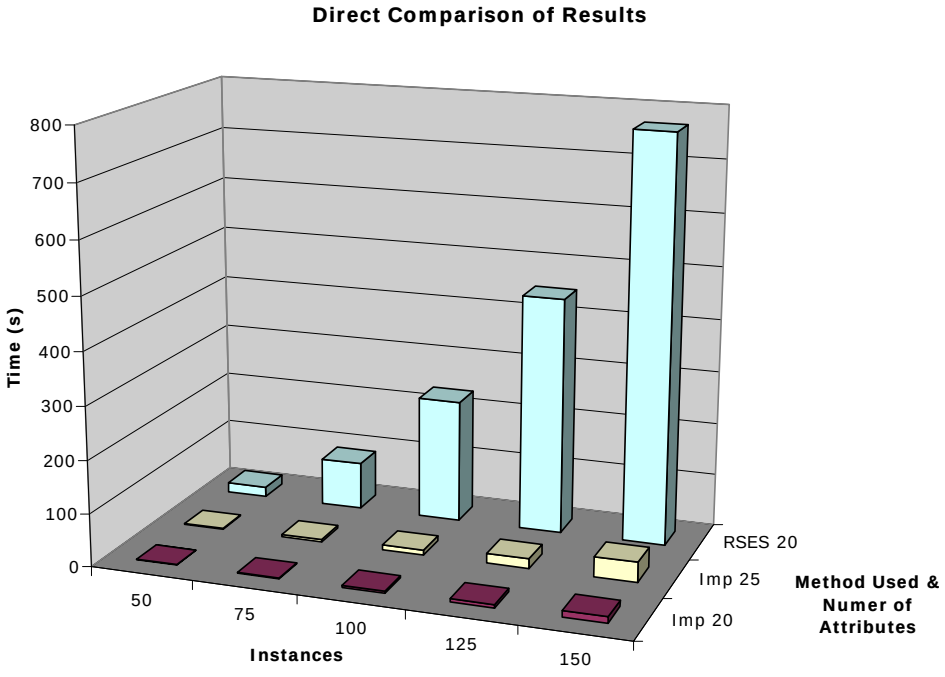

Figure 2.4 – Direct comparison of RSES results vs. the improved algorithm results.

The computation time for this program never went above a thousand seconds for the data tested. There are several possible reasons for this rather significant difference. RSES is a Java program, which may contribute to its lack of speed. However, considering the magnitude of difference, even a 3x speedup would not bring it up to par with the algorithm reported here. In order to get a better idea of the level of improvement this algorithm provides a few additional tests in the 30 & 40 attribute 2 class data sets with rows extending out to 500 instances were done. Results are as shown in Table 2.11.

Table 2.11 – Reduct generation test for larger data sets.

Attributes	30	40
Instances	Time(s)(#reducts)	Time(s)(#reducts)
50	2.9375 (5723)	9.2969 (18630)
75	9.9063 (14023)	33.8281 (46825)
100	21.0625 (15432)	77.0313 (54109)
125	48.9063 (18739)	197.3438 (70757)
150	109.7031 (29812)	457.4375 (119554)
200	334.3125 (60707)	1578.3 (271062)
300	1124.0 (73148)	5969.8 (348187)
400	3394.3 (83846)	20639 (465852)
500	9117.3 (153687)	59945 (976120)

At the 500 instances and 40 features point the improved algorithm has a computational time of 59945 seconds, which is comparable to the 50400 second time that was needed while using RSES at 75 rows and 30 attributes. This is a significant improvement in overall performance.

2.10 Conclusions

Significant computational time improvements in exhaustive reduct generation have been accomplished. By using a binary discernibility list with several modifications to the reduct generation algorithm, a substantial improvement in the time needed to generate all reducts for a given set was obtained. However, the problem sizes that can be handled by this method are still small compared with those of practical interest (i.e. large databases). A more real world problem size of 100 attributes by 10,000 instances, with a time estimation based on the data that has been collected would yield a computation time in terms of decades or possibly hundreds of years, too great a time to be practically useful. A more accurate estimate is not possible since this type of problem has been proven to be NP-complete. A database of 500 instances with 40 attributes required over 16 hours to complete, even with the developed improved computational method, so it is no great stretch to assume that a 100-attribute problem would take significantly longer. For the generation of larger data set's reducts to be feasible, either precision must be sacrificed or much more computational power must be levied against this problem or a still better reduct generation method developed.

It is known that reducts are useful for generating accurate rule sets for classification; as such an efficient method for generating reducts for large datasets would be very useful. Furthermore, the use of discerns instead of reducts should also be considered because of

their greater robustness and redundancy. The next chapters continue to investigate means of reducing computational time while maintaining classification accuracy with the goal of creating a better classification method for use with large datasets.

3. Generating Rules from Classifiers

3.1 Introduction

Up to this point, only the generation of reducts has been discussed. However, to properly evaluate the quality of the results, it is necessary to find a way to determine the quality and related classification performance of the generated reducts. This chapter presents an overview of methods that use reducts to generate classification results, and presents in detail the algorithm used to produce classification results. In particular, this approach is needed to test the quality of the statistical reduct generation proposed in Chapter 4 and to have a fair comparison of the reduct generation method introduced in Chapter 2 to the method proposed in Chapter 4.

3.2 Rule Generation

For reference, a rule is considered a set of specific feature values that are present in several instances in the data set and can be used to correctly classify signal instances. The simplest and most desirable rule consists only of a single feature and its associated value. Consider the Section 2.6 reduct example. An example of a rule for this data set would be; when feature 1 equals 1 the classification is set 1. This rule is observed easily enough by

looking at Table 2.2 or Table 3.1. A full explanation of how this and the other rules were extracted for the dataset is presented in Section 3.4.1.

General rules with high set coverage are of the most desirable type because they allow for the easiest classification/characterization of a data set. They also represent the most prevalent features that distinguish a given set of data points from the rest. Simpler rules, in general, capture the dominant characterization of large numbers of data elements, while complex rules characterize smaller sets of points that require more specific characterization. This observation follows the Kolmogorov complexity definition [Gam99] that predicts longer computations for more complex problems. To some degree, rules that capture the behavior of very few data points correspond to basis functions approximating local errors. As local errors get smaller after a dominant basis function is applied to approximate an unknown function value, many more basis functions may be needed to reduce local errors, which in turn may lead to over fitting. Thus, analyzing function approximation using too many specific rules that apply to very few data points, may lead to statistically unstable characterization of the object properties of data points (like determination of class ID).

There are many possible rule generation algorithms. Chapter 6 in Ohrn's thesis [Ohr99] gives a basic introduction to rule generation methods. A good example of a rule generation algorithm known to produce good results is the a priori based algorithm discussed along with several other known algorithms in Agrawal's work on fast

generation of association rules [Agr94]. A discussion of the efficient implementation of the a priori algorithm can be found in Borgelt's work [Bor03]. The rule generation algorithm used in this work is not of the quality of many of the alternatives. However, it has the advantage of being simple and relatively easy to implement, something many of the alternative algorithms are not. The rule generation algorithm itself is covered in the following section.

3.3 The Rule Generation Algorithm

In the following demonstration, the method used to determine rules is as follows:

I. Staring with a data set and a set of generated discerns, randomly choose a discern from the set of known discerns.
II. Using an ascending row sort, sort the discern's associated data (columns associated with the discern elements).
III. Scan the reduct/discern for rules.
    a. Check for similarities between rows.
    b. Start with the simplest rule (i.e. a single feature value) and scan the sorted rows until there is a discrepancy in the decision value, or the rule correctly classifies all the data rows it covers.
    c. If there is a discrepancy (as determined in b), go back and try a more complex rule (by adding another feature to the rule).
    d. A rule must hold true for a least 5 rows, otherwise, it is considered too specific, and should be discarded.

e. When a rule is discovered, compare it to preexisting rules to ensure that some other better rule does not already exist to cover it.

IV. After every *n* (A typical value of *n* can range from 10 to 100, although, 20 seems to work well.) rules are discovered, resort the rule list with the simplest rules at the top. This will allow for faster completion of part IIIe. When a new rule is found that either already exists or is covered by one of the earlier rules, it will be discovered more quickly when the more general, simple rules are at the top of the rule list.

V. Rotate the discern columns to generate better rule coverage and repeat Steps II-IV, i.e. columns {1,2,3,4} become {2,3,4,1}.

VI. Repeat Steps I-V until all discerns are used or rule discovery flattens out. (Experiments have shown that rule growth is proportional to the square root of the number of reducts/discerns.)

3.4 Examples

Classification is an important part of many applications from financial analysis to biomedical scanning applications. As has already been discussed, reducts, and by association, discerns, can be used to classify data. How this is done, however, has not yet been discussed. This section will provide a pair of examples illustrating the use on a set of reducts of the above rule generational algorithm and how the resulting rules are able to generate classification results.

3.4.1 Random Example

In this section, the results from the example in Section 2.5 are taken and expanded upon to show how the discovered reducts of the original data can be used to generate rules by which the data can be classified. The reducts associated with the data set from Section 2.5 were $\{1,3\}$ & $\{1,4\}$. Let us begin with the $\{1,3\}$ reduct. Thus, the first part of this example illustrates rules based on this reduct only. Next, as stated by Step II, columns 1 & 3 of the previously labeled data (as taken from Table 2.3) are taken and sorted as shown in Table 3.1 below.

Table 3.1 – Sorted data for columns 1 & 3.

Sample	Feature 1	Feature 3	Classification
9	1	3	1
1	1	5	1
6	2	5	3
4	3	3	2
10	3	3	2
8	3	5	1
5	4	1	2
2	4	5	1

Next is the execution of Step III. This is the most important step in that it is the one that actually extracts rules from the data. In order to "Check for similarities between the rows," the process begins at the first row of the sorted data with an assumed initial rule of: "If Feature 1 equals '1' then the classification is '1'. The algorithm must then check to see if classification is consistent for all such instances where Feature 1 equals '1'. Since that is the case here, the first rule can be considered to be found. The process then

continues to the third row, where it is determined that a value of ‘2’ for Feature 1 implies a classification of ‘3’, and so forth. The minimum coverage rule of step 3-d is currently being ignored. It is, in fact, a mostly arbitrary setting and is dependent on the desires of the user. Too large a value may result in too few rules discovered, while too small a value may results in a large number of rules that apply to only one or two signals. Nevertheless, the following rules can be derived from the above table:

Table 3.2 –Set of discovered rules for 1& 3.

Rule	Feature 1	Feature 2	Feature 3	Feature 4	Classification
1	1				1
2	2				3
3	3		3		2
4	3		5		1
5	4		1		2
6	4		5		1

However, this is not the end of the process; after the first iteration, the reduct is rotated so that the columns 3 & 1 are observed in the stated order. This is followed by rule generation based on the second reduct and any resulting rotations thereof. Not all permutations of longer reducts are examined, since doing so would be computationally prohibitive. The rotation scheme provides a balance between computation time and coverage (*n* operations vs. *n!* operations). Under normal circumstances when dealing with a larger database with more features, there would likely be many more than two reducts to examine. Under those conditions, the process of generating rules would continue until all the reducts were examined or a plateau was reached in the generation of rules. The set

of all possible rules for the data set being examined is displayed below in Table 3.3. The first six rules are identical to Table 3.2; the next set of 7 rules arise from the rotation of columns 3 & 1; columns continue to be rotated for longer reducts and for subsequent reducts until all possible rotations are exhausted or until some stopping criterion has been reached.

Table 3.3 – Set of all possible reduct based rules.

Rule	Feature 1	Feature 3	Feature 4	Classification
1	1			1
2	2			3
3	3	3		2
4	3	5		1
5	4	1		2
6	4	5		1
7		1		2
8	1	3		1
9	3	3		2
10	1	5		1
11	2	5		3
12	3	5		1
13	4	5		1
14	1			1
15	2			3
16	3		3	1
17	3		5	2
18	4		1	2
19	4		2	1
20	4		1	2
21	1		2	1
22	2		2	3
23	4		2	1
24	3		3	1
25	1		5	1
26	3		5	2

Now let us eliminate the duplicate and redundant rules. For example, rules 14 and 15 are duplicates of rules 1 and 2. And rules 11 and 22 are already covered by the simpler rule 2; rule 2 is shorter but equal to rules 11 and 22 for the feature they have in common and has the same classification value, therefore, rule 2 is said to cover rules 11 and 22 because it would classify the same pieces of data as rules 11 and 22. The resulting set of rules would be as shown in Table 3.4.

Table 3.4 – Reduced set of all possible reduct based rules.

Rule	Feature 1	Feature 3	Feature 4	Classification
1	1			1
2	2			3
3	3	3		2
4	3	5		1
6	4	5		1
7		1		2
16	3		3	1
17	3		5	2
18	4		1	2
19	4		2	1

As can be seen from Table 3.4, the set of all possible rules has been reduced from 26 rules to ten rules. During normal execution of the above algorithm, this reduction of rules would happen on a continuous basis, as it is more computationally time efficient to handle rule generation in that manner. For example, Rules 8, 10, 13 & 15 in Table 3.3 would not even be stored in the set of rules since they either already exist in the set or are already covered by a simpler rule. Rules like Rule 7, however, are by necessity handled

differently. Once a rule, for example, Rule 7, is discovered, it is passed off to a separate function which compares it to other rules to see if it covers any existing rules, as happens to be the case here with respect to Rule 5. Normally none of these rules would meet the minimum coverage criteria mentioned earlier, but because of the extremely small size of this data set, it is necessary to bypass that criterion. Under normal circumstances, there would be hundreds and even thousands of possible rules and the coverage criterion helps limit the number of rules to a more useful and manageable number.

Now that the set of rules has been discovered, the rule set needs to be applied to the original data set. The following table contains the relevant features and the vote count for each classification based on the application of all the discovered rules. Each time a rule is satisfied for a particular data point, the vote count is increased for a particular class associated with this rule. In the example data set each data point was covered by one or two rules. In this case the testing data is the same as the training data, and while this test achieved 100% coverage, this won't necessarily always be the case. Under other circumstances, the row coverage rule would not be waived and there would likely be elements that are not covered due to the row coverage limitations.

As seen in Table 3.5, all the results, when compared to their correct classifications, are correctly classified with no conflicts or incorrect classifications. Under normal circumstances the entire data set would not be used to discover all the potential rules, thus resulting in incorrect classifications. This latter situation occurs when rules are developed

based on training data and then applied to a different set of testing data. There will be instances where rules may apply to two or more different classes. In such instances, the "winner" is typically the class to which the most rules apply, as it is the most statistically likely case.

Table 3.5 – Rule based vote count.

	Feature values				Vote count		
Sample	Feature 1	Feature 2	Feature 3	Feature 4	Class 1	Class 2	Class 3
1	1	4	5	2	1		
2	4	1	5	2	2		
4	3	2	3	5		2	
5	4	4	1	1		2	
6	2	4	5	2			1
8	3	2	5	3	2		
9	1	5	3	5	1		
10	3	3	3	5		2	

3.4.2 The Iris Example

The following example makes use of the well-known Iris database [Fis88] and was chosen because of its widespread exposure and relative simplicity.

The Iris database has 4 reducts in its non-scaled form. It is considered unnecessary to scale/label the database because the data is already limited to a relatively few discrete values. The reducts for the Iris database are: $\{1,2,3\}$, $\{1,2,4\}$, $\{1,3,4\}$, and $\{2,3,4\}$.

To get some idea of the effect of step 3d on the generation of rules and the end classification results, examine the results using the default value of 5 rows minimum covered by a rule. Table 3.6 shows all rules discovered using the rule generation algorithm and the four reducts previously mentioned. Applying the aforementioned rules against the Iris database results in a correct classification rate of 78.9%. This low percentage is a result of the database having several rows that are unique enough to escape coverage by rules that cover a minimum of five rows. It is also a function of the overall size of the training data set. In this case, 5 data points was 3.4% of the whole training set.

Table 3.6 – Iris database rules.

Rule	Feature 1	Feature 2	Feature 3	Feature 4	Classification
1				0.2	1
2				1.3	2
3			1.4		1
4			1.5		1
5				2.3	3
6			1.3		1
7			1.6		1
8				0.3	1
9				0.4	1
10				1.0	2
11		3.5			1
12			5.6		3
13				2.0	3
14				2.1	3
15			4.0		2
16			4.7		2
17				1.2	2
18			4.5	1.5	2
19	4.8				1

Decreasing the coverage minimum will increase the amount of time it takes the rule generation algorithm to run; however, it will also improve the classification performance. How much it improves the classification depends largely on the database being tested. The Iris database for example, did not yield a high “correct classification” at the default coverage setting of at least 5 rows per rule. The following Table 3.7 shows the effect of a range of coverage values on classification results for the Iris database.

Table 3.7 – Coverage values vs. correct classification of Iris database.

Minimum Coverage	Num or Rules	Correct Classification %
1	224	100
2	76	95.92
3	44	93.88
4	27	86.39
5	20	79.59
6	14	72.11
7	10	65.31
8	6	60.54
9	5	55.78
10	5	51.02

The above table makes it apparent that the number of rules found increases rapidly at low coverage values, which for this database, is at coverage values below five. For the Iris database the value of using rules is obscured at these low coverage values. At reasonable coverage values, the correct classification percentage is relatively low. However, for larger, more complex databases, I have not observed this to be the case. More typically,

an appropriate minimum coverage value can be found that gives a reasonable percent correct classification.

The previous examples dealt with the use of the full dataset for both testing and training operations. Table 3.8 depicts a series of results using various training/testing percentages with varying minimum coverage values. In the first instance, for example, ninety percent of the database is used to train the algorithm (generate reducts and rules), while the remaining ten percent is tested against the results.

Table 3.8 – Train/Test ratio vs. correct classification of Iris database.

Train/Test Ratio	Minimum Coverage	Num or Rules	Correct Test Class. %
90/10	1	202	86.67
90/10	2	72	80.00
90/10	5	17	73.33
80/20	1	151	86.21
80/20	2	63	75.86
80/20	5	15	65.52
60/40	1	137	91.53
60/40	2	52	81.36
60/40	5	10	66.10

The results of the above table give classification results that are below those seen in table 3.7, but that is expected since not all the database is being used to generate the classification rules. However, the trends in the classification results of the Table 3.7 remain true in Table 3.8.

Another reason to limit the number of rules generated by limiting both the number of reducts scanned and setting a minimum coverage point is that too many rules can lead to over training in addition to greater computational time. Over training is undesirable because it results in diminishing returns. An algorithm becomes too focused/specialized on the training results so that it is unable to handle new data as well as it did the original training set. The situation can be said to be similar to over specialization.

3.5 Closing Remarks

As seen by the preceding examples describing the applied rule generation algorithm, it is a simple, but not the most efficient rule generation. However, its simplicity makes it easy to implement and its performance is good enough to match better algorithms when greater time expenditure is acceptable. This rule generation algorithm is used in Chapter 4 for both the exhaustive method for generation of discerns and a statistical method introduced in Chapter 4 due to its simplicity and ease of implementation.

4. A Statistical Approach to Discerns in Classification

4.1 Introduction

This chapter presents a thorough examination of a heuristic statistical approach to finding classifiers for use on knowledge databases. The exhaustive methods examined in Chapter 2 have proven to be too time consuming as a result of their NP-complete nature. Thus, this chapter focuses on other less precise, but hopefully just as effective, methods for producing classifiers. The overall goal of Chapter 4 is to present a simple, yet effective, method for producing good classifiers in a reasonable amount of time using rough sets. In the following sections a new statistical method for finding classifiers based on reducts and discerns will be introduced. The organization of this chapter is as follows: first, is a short discussion of prior research and other relevant work; second, is an explanation of the statistical techniques used in this paper and how they are applied; third, is a discussion of the algorithm used for the presented research; and lastly, the results and the discussion are presented.

4.2 Prior research on partial reducts

In Chapter 2 an accelerated exhaustive reduct determination algorithm was examined, which, despite being faster than other exhaustive methods examined, was unable to handle large database problems in an acceptable amount of time. This work reports an even faster, but less exact method, of finding reducts and discerns for use in classification. There are several other existing algorithms used to find partial sets of reducts, one of the best known of which is the genetic algorithm. What follows are a series of brief explanations of existing methods for generating partial reducts and/or partial sets of reducts.

4.2.1 The Genetic Algorithm

The genetic algorithm has a history dating back to the 1970's when it were first introduced as an algorithmic optimization method by John Holland [Mel96]. Genetic algorithms are one of the most widespread data mining and extraction algorithms in use, with applications including reduct generation, floor planning, behavior learning, circuit design, code breaking, and many more. Wroblewski [Wro95] presents an explanation of how to use genetic algorithms to find minimal reducts. Vinterbo gives a good explanation of the genetic algorithm in his thesis work [Vin99], as well as some useful references for additional information.

A genetic algorithm searches the space of all potential reducts, by iteratively refining the fitness measures of a set of potential solutions (the population). The fitness measure is defined using the fitness function values of the individuals within the selected population. The fitness function quantifies the optimality of an individual solution (chromosome in genetic algorithm terms). The population may be randomly generated or determined using some heuristics. Populations are refined by "mating" pairs of parents to generate more viable offspring to replace the parents. If desired, "mutations" or random deviations in the chromosome can be introduced. In the simplest form of this "mating" operation, also called a *crossover*, a random position is chosen in each parent reduct/chromosome for splitting, and the resulting offspring are assigned one section from each parent. The simplest form of mutation is implemented by simply flipping a random bit in the individual. The selection of which parents undergo this operation is stochastic based to support the propagation of fitter individuals, thus "evolving" more effective results.

4.2.2 Set Covering Heuristics

Set covering heuristics are various heuristic methods than can be used to cover a data set. Heuristics are best described as rules used for processing information on a rule of thumb. A heuristic algorithm is one that has a provably good run time with near optimal solution quality. David Johnson presented the basis for set covering heuristics in his 1974 paper [Jon74]. Set covering itself is very similar to the process used to find reducts. By definition, the set covering problem (SCP) consists of several "sets" with some common

elements with the goal of selecting a minimum number of sets so that the sets picked contain all elements that are contained in the input set. It should be clear how similar this definition is to the reduct generation problem, since it also requires the selection of a minimum number of features with all the elements from the full set still being represented. All of the mentioned algorithms make some use of heuristics. In fact, the algorithms proposed by Hu [Hu03] make use of a feature ranking method very similar to the prioritization method mentioned in Chapter 2.

4.2.3 Approximate Hitting Sets

A hitting set is a (non-empty) set that intersects with all sets in a collection of sets. An approximate hitting set is a hitting set that intersects with some fraction ***r*** of all the sets. For approximate hitting sets, Vinterbo's thesis [Vin99] is a good source of background and other information. With regards to reducts, ***r***-approximate hitting sets are essentially identical in practice to approximate reducts. There are also similarities to both dynamic reducts and genetic algorithms.

4.2.4 Dynamic Reducts

Dynamic reducts have seen significant attention from Bazan with research results presented in several of his papers. For example, Bazan introduced dynamic reducts as a way to produce better rules from decision tables [Baz94]. Later, as part of a book on rough sets, he presented a comparison of dynamic and non-dynamic methods for

extracting rules [Baz98]. Dynamic reducts themselves, exist as part of a response to the sometime chaotic nature of the data under analysis. They provide better accuracy in situations with noisy and missing data. Because of their flexibility, they can often provide better results with “unseen” data then do normal static reducts. Dynamic reducts are typically generated by subdividing the “training” data into several tables and then finding their associated reducts. The reducts common to these sets are then referred to as dynamic reducts. A reduct need not be present in all the subsets, but must be in some suitable number of them. It is thought that since these reducts are common to subsets of the training data that they will also likely be present in any testing data, thus allowing for better overall classification.

4.2.5 Ensemble Systems

Ensemble systems are arrangements of two or more classification systems into a single system. They work by combining the results of individual classifiers to provide more accurate results. The work by Robi Polikar [Pol06] provides an extensive overview of ensemble systems, including their background, implementation, and some of the potential classification systems that can be used to create such a system. According to Polikar, the process of using an ensemble system is similar to asking several doctors’ opinions and then weighing the individual results against certain criteria to determine which results to choose. Within the paper, several well-known ensemble algorithms are discussed, including bagging and its variations, the AdaBoost algorithm, and other methods. Also

discussed is the system by which the ensemble weighs the results from its different components and tries to choose the best solution.

4.2.6 Other Methods and Research

The methods mentioned in this paper so far only examine the tip of the iceberg. For example, Ras and Dardzinska [Ras06] presented an efficient algorithm for producing rules from “incomplete information systems.” In addition, Ras and Im [Im05, Ras07] have been researching ways of keeping confidential the relationships between data and the classification results. The relevance of this work is shown with regards to importance currently placed on keeping records private. To accomplish this, they have devised a method to decouple the results of a data mining operation from the original data. Another example of the application of rough set theory via reducts is the work by Tseng et al. [Tse05] concerning the application of reduct based rule generation to determine rules for classifiying machined surfaces for quality assurance purposes. Additionally, work has been done with regards to the characterization of partial reducts and their complexity [Mos05]. More recently Moshkov et al. [Mos07] have done some work that evaluates the creation of partial reducts via several different algorithms and looks at the quality and quantities of partial reducts generated for various algorithm configurations. There exist many other methods and variations with which to generate and use reducts and rules. However, to cover these is beyond the scope of this work.

4.3 How Reducts Are Found Statistically

As described in Chapter 2, exhaustive reduct determination methods become unusable as data set sizes grow beyond the small to medium size range. (E.g., Databases with less than 20 features and 500 instances can be effectively managed using exhaustive reduct methodology.) Previously, a random dataset with 40 features, 500 instances and 2 classes was tested. The time to exhaustively find all reducts using that improved algorithm was 59945 seconds or roughly 16.5 hours. Imagine then, knowing that the growth in time is approximately exponential, how long it would take to analyze a larger database.

The approach reported here uses statistical methods, (e.g. statistical sampling techniques) to reduce the work needed to find reducts. Thus, the research effort should be concentrated on finding a few minimal reducts, specifically the reducts that provide the best information for rule generation. Three possible routes were evaluated. The first approach was to simply sample the original data set. The second method was to generate a discernibility matrix and then "sample" it by randomly selecting instance pairs to compare over the whole dataset. The third approach examined was to create a statistically sampled version of the full discernibility matrix, determine which features are the most common discerning features, then remove them to make them core features and reduce the sampled discernibility matrix size. However, removing features this way creates discerns rather than reducts. Discerns were introduced by Starzyk [Sta00] and are a convenient way to express information system properties. In this thesis, they will result

from developing reducts on a subset of the binary discernibility matrix. A discern is a subset of features as is a reduct; however, a discern is not a reduct because in many instances, it can still be reduced in size and maintain the discernibility relations between all data elements.

The first method, sampling the dataset, is simple but sacrifices accuracy for speed. It is the same as running the exhaustive test on a subset of a database as is done later in the paper for comparison to the hybrid statistical method. These subset tests take a portion of a dataset for training and use the remaining portion for testing. Using this sampling method may be satisfactory if high accuracy is not required, but the method presented in this paper will generate results of near equal quality with the reduced data set even faster. Furthermore, to get a manageably sized data set from a truly large set would require a vast reduction in sample set size, thus, sacrificing accuracy as discussed later in the paper (Section 4.5).

The second method, generating a discernibility matrix and sampling it, also sacrifices accuracy and would still be too slow. While it would not require the creation of the full matrix to generate the sampled discernibility matrix, thus side stepping the memory issue with large discernibility matrices, an accurate representation of the full discernibility matrix would still take too long to process for reducts because the time needed grows exponentially with discernibility matrix size. For example, to get 95% ± 0.5% accuracy

on a 10,000 element discernibility matrix, the sample matrix would need to be at roughly 7900 elements, a value that would still be too large to process in a reasonable time.

There are two useful ways to implement the third method. One is to rely on a statistically reduced version of the sampled discernibility matrix to find reducts (or discerns), which will be referred to as the **standard statistical method**. The other, referred to as the **hybrid statistical method**, generates a reduced discernibility matrix from the sampled one and uses the features removed to produce a version of the full discernibility matrix of the approximately desired size.

The standard statistical sampling method, while slightly less accurate, is applicable to all sizes of databases, while the hybrid method is more costly time wise due to the necessity in generating a discernibility matrix from the full data set. The faster standard statistical method would statistically sample the discernibility matrix and determine the most common differentiating features. Those highly differentiating features would become core features of any resulting “reducts,” obtained from the reduced sampled discernibility matrix. The reason to use these new “core” features is reduction of the discernibility matrix to a manageable size.

The hybrid statistical method, using the same sampled discernibility matrix, also reduces the sample matrix by selecting the most common differentiating features. However, it uses the reduction in size to estimate the resulting full discernibility matrix size with the

selected feature(s) removed, and continues to remove features until the estimated full discernibility matrix reaches a desired size. This method then generates the full discernibility matrix with all rows containing the pre-selected features removed, and has the advantage of creating a 100% accurate discernibility matrix preserving all the original data associations. The downside is the time required to process the full discernibility matrix.

In both statistical sampling methods the generated reducts can not be verified as actual reducts and must be considered as discerns of the original information system as previously mentioned. This thesis will focus primarily on the hybrid method for generating results. However, a comparison between the hybrid and standard statistical methods will be presented in Section 4.6.

As noted, each method has its pros and cons, but before the selected hybrid statistical method is discussed in greater depth, it is necessary to describe how the sampling process works.

4.3.1 Population Sampling

First, it is necessary to determine the size of the sample needed from the “full” discernibility matrix to get a smaller but accurate representation of the full set. This will save memory and greatly reduce processing time. Determining the appropriate sample

size requires three things: a genuine random sampling of the population, the setting of a desired confidence level, and the setting of a confidence interval. The confidence level is the measure of certainty, while the confidence interval describes the range of values in which the result falls. For example, let us assume that through sampling that it has been determined that Feature 4 is present in 62% of a given discernibility matrix's entries. A confidence level of 95% with a confidence interval of 1% implies a surety of 95% that the result falls between 61%-63%. Given the confidence level, confidence interval, and the population size, the following equations are used to determine the sample size needed to meet the specified confidence values.

Population size refers to the number of signal pairs in discernibility matrix

$$P_{size} = \frac{n(n-1)}{2}, \text{ where } n \text{ is the number of data points} \tag{7}$$

In statistical determination of features that differentiate signals in the discernibility matrix, the confidence is level defined as:

$$Con_{Lev} = 1 - \int_{-\infty}^{Z} \frac{1}{\sqrt{2\pi}} e^{\frac{-u^2}{2}} du \tag{8}$$

and related confidence interval to determine the minimum required sample size as follows:

$$S_{size} = \frac{\eta}{1 + \frac{\eta - 1}{P_{size}}}, \text{ where } \eta = \frac{Z^2 * 0.25}{Con_{Int}^{\ 2}} \tag{9}$$

Example:

Let us consider a 10,000 element discernibility matrix and apply population sampling. To find the required sample size such that the resulting sampled matrix will match the full discernibility matrix by at least 99.0%, S_{size} is calculated from (7) as follows:

$Con_{Lev} = 99\% \Rightarrow Z = 2.576, Con_{Int} = 1.0\%,\ P_{size} = 10000,\ \eta = \frac{2.576^2 * 0.25}{.01^2} = 16587,$ and

$$S_{size} = \frac{16587}{1 + \frac{16587 - 1}{10000}} = 6239$$

This example shows that given a 10,000 element discernibility matrix, a set of 6,239 of the full discernibility matrix's elements need to be sampled to get the required results with a 99.0% confidence. The resulting set is still fairly large, but it should be noted that the amount of savings increases exponentially with the size of the sample set. (Z is calculated from equation 2 or can usually be found in lookup tables.)

4.3.2 Reducts and Discerns in the Sampling Method

Of the three potential methods mentioned above that use sampling, the third method, the hybrid sampling method, was chosen for this chapter as it was expected to give the desired combination of low computational time and high accuracy. This method reduces the size of the discernibility matrix to significantly reduce the search space before finding reducts, thus reducing the time needed, while maintaining an accurate representation of the knowledge data. The process occurs as follows:

1) Given *n* data elements calculate the maximum possible discernibility matrix size.

$$P_{size} = \frac{n(n-1)}{2}$$

2) Randomly select pairs from the data set to create randomly sampled discernibility entries. (After making sure they are not of the same classification.) Sample the number of pairs as determined by the S_{size} equation (7). For instance, a data set of $n = 10{,}000$ signals, with the same confidence level and interval as the above example, would result in a maximum discernibility matrix size of 49,995,000 rows. Thus, by the S_{size} equation,

$$S_{size} = \frac{16587}{1 + \frac{16587 - 1}{49995000}},$$

a 16,582 row sample discernibility matrix would need to be created from randomly selected pairs of data from the original data set to obtain a 99% accurate representation of the full discernibility matrix.

3) Count the feature representation throughout all rows of the sample discernibility matrix and select the most common feature.
4) Remove all elements of the discernibility matrix containing the selected feature.
5) Repeat steps 3-4 until either the maximum number of selected features has been reached or the estimated discernibility matrix has reached the desired size.
6) Using the selected features, build the "true" discernibility matrix based on the full data set. With the selected features removed, the resulting discernibility matrix should be of an easily handled size.
7) Find all possible discerns using the new matrix.

One of the major advantages of the statistical method, aside from the time savings, is that the sample discernibility matrix size will always be constrained by the η calculation value. No matter what value the maximum discernibility matrix size, P_{size}, takes, the

sample discernibility matrix size determined by the S_{size}, equation will never exceed the value determined by the calculation of η. Meaning, that in 2), above, S_{size} will never exceed 16,587.

However, the major disadvantage of this statistical method is that it actually generates discerns as opposed to reducts. In addition, once the features are removed and the reduced discernibility matrix is built, there is no way to compare the results against the removed features to see if a discern or a reduct has been generated. Testing each discern against the full data set, without building a discernibility matrix, requires several $O(n^2)$ operations (up to the number of redundant features). Therefore, the resulting sets are classified as discerns because it would be too computationally intensive to pick out the true reducts or to convert discerns into reducts.

Testing of several smaller datasets shows that the quality of the results varies rather drastically, from 0% actual reducts to 100%, and seems to be dependant both on the size of the dataset examined and its composition. Table 4.1 illustrates the accuracy of this method using the mushroom database [Mur90], by showing what percent of the generated results were actually discerns as opposed to true reducts. For example, the first row shows results for a trial in which 5% of the ***mushroom*** database was used for learning, meaning the selected fraction is what is used to find the discerns. The program then ran for the specified number of iterations, and each time the discerns were checked to see if they could be reduced further to true reducts or were already reducts. The “Average %

Reducts" measurement specifies the average percentage of discerns that were true reducts. Ideally, one would like the "Average % Reducts" to be 100%. The actual percentage results vary from database to database. The disadvantage of having discerns instead of reducts is that the subsets are larger than true reducts. On the other hand, discerns tend to provide greater redundancy and possibly better rules.

Table 4.1 – Sampling for reducts.

Learn/Test Ratio	Iterations	Average % Reducts	Std Dev % Reducts
5/95	10	31.32	10.43
10/90	10	39.32	6.55
20/80	10	50.74	23.69
30/70	10	17.43	11.25
40/60	10	21.96	19.81
50/50	10	14.37	16.51
60/40	10	7.38	11.67
70/30	10	7.53	12.20
80/20	10	0	0.00
90/10	10	0	0.00

Figure 4.1, provides some idea of the accuracy in the sampling method by showing how the sampled results degrade in quality with subsequent feature selections (iterations). The following graph illustrates the percent difference in the normalized feature counts of the sampled discernibility matrix vs. the full discernibility matrix. In other words, if Feature 2 is present in 90% of the full discernibility matrix's entries, and present in 88% of the sampled discernibility matrix's entries, there is a 2% percent difference between their

normalized feature counts. For each method, three iterations were performed counting features in the both full and sampled discernibility matrices. To clarify, each iteration removes a feature, then re-determines the remaining features' representations within the sampled and full discernibility matrices, and then determines the differences between the two sets of results.

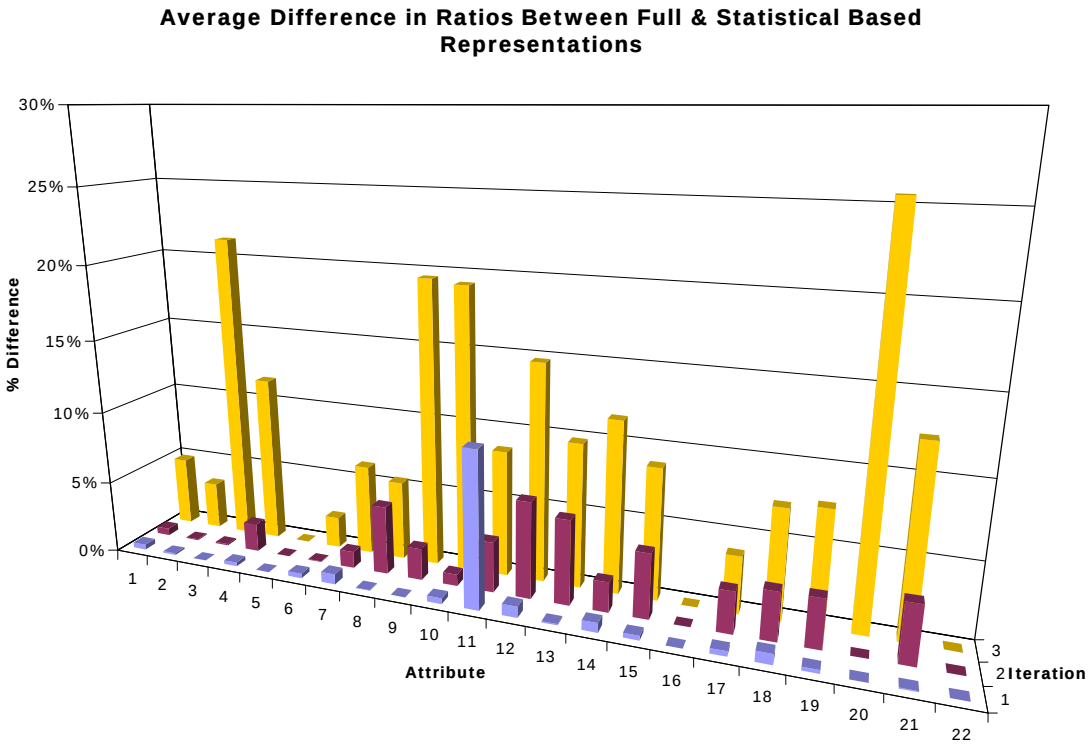

Figure 4.1 – Column count difference vs. feature selection iteration.

The chart shows the percent difference for each feature after each iteration, i.e. the counts are normalized and the calculation is done by subtracting the statistical percent from the normalized exhaustive percent. Initially, the difference between the two methods is near

zero, but tends to exceed 5% by the third feature selection, meaning that the statistical representation becomes less accurate as the number of iterations increases (i.e., as more features are removed). What this means is that as features are selected using the sampled discernibility matrix and elements are removed, the sampled matrix deviates increasingly from the full discernibility matrix. If the process continues long enough there will be instances where a selected feature is present in ~90% of the sampled matrix, but would only be present ~50% of the full matrix. Such a case would result in longer computational times later in regard to Step 6 in Section 4.4, because fewer than expected rows would be removed resulting in a larger matrix and greater computational time. The opposite can also occur, resulting in too small a matrix and fewer than expected discerns.

4.4 The Hybrid Statistical Reduct Determination Algorithm

This section will give a detailed overview of the full process, paying particular attention to the parts that differ from the algorithm examined in Chapter 2 (see Figure 2.1). The actual reduct generation algorithm remains the same with the exception that Step 3 of the previously mentioned algorithm (Section 2.4) is replaced by Steps 3-6 below. The additional steps are present because the algorithm is now used for classification and not just the generation of reducts. Figure 4.2, following the algorithm description, presents a flowchart of the replacement steps for Step 3 of Figure 2.1 (Discernibility Matrix Generation), showing how steps 3 through 6 below fit into the original algorithm. Steps 7

and 8 summarize steps 4 through 12 of the previous algorithm, while the remaining steps (9-11) represent the remaining rule generation and classification requirements.

The algorithm operates as follows:

1) Discretize, scale, or label the data as needed.
2) Partition the data by randomly selecting elements for the training/testing sets.
3) Build a randomly sampled discernibility matrix whose size is determined based on the desired confidence level, confidence interval, and maximum discernibility matrix (population) size.
4) Determine the most common features, and use them to reduce the discernibility matrix to a manageable size (usually around 500-1000 rows) or until 15% of the total features are selected, whichever comes first. This is accomplished by removing rows of the matrix that contain the selected features. How this was done was discussed in the previous section on population sampling (section 4.3.1). Figure 4.1 illustrates how the sample Discernability matrix begins to diverge from the full Discernibility matrix as features are removed.
5) If the data set contains more than a certain number of features, truncate it at a desired point so that only the most common features are examined.
6) Build the full discernibility matrix with the rows containing the selected features removed. This should result in a discernibility matrix size of roughly the same size as was estimated during the feature selection. (During this process, the data can be examined for any core features that are not already in the selected feature set.
7) Remove rows from the reduced discernibility matrix that contain any new core features discovered in Step 6.
8) Find all possible discerns using the reduced discernibility matrix and the selected features as the core.

9) Determine Rules: Randomly selecting discerns, find rules for the data set. (Each rule must cover a certain number of rows, for example, 5.) To avoid searching all discerns, stop searching once the rule generation levels out.
10) Apply the discovered rules to the previously generated random test data set, to determine their classification accuracy.
11) Go to Step 2 for repeated iterations.

Now, the various steps of the algorithm will be examined and additional explanations given. Step 1 is more complicated than it might seem at a first glance. The preparation of data is one of the most important parts of the entire algorithm, but can vary greatly depending on the end application and the results desired. However, whatever the case, the data needs some form of preparation before it can be used, from a simple linear scaling to a full statistical analysis and transformation. An extensive guide to data preparation for use in data mining and similar operations is provided by Pyle [Pyl99].

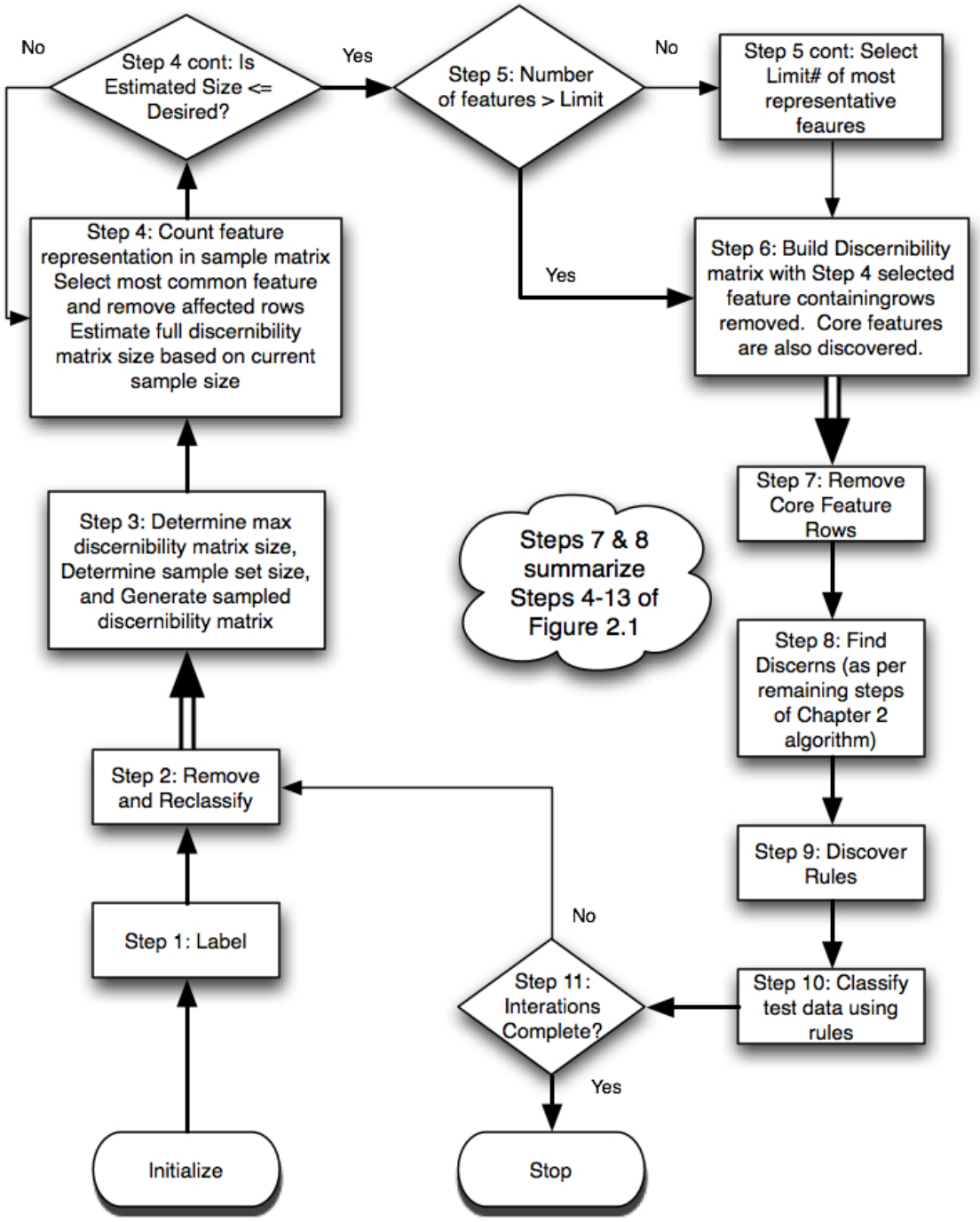

Figure 4.2 – Flowchart of changes to the reduct algorithm.

Step 2, manages the partitioning of the data set. When testing it is important to set aside a portion of data for testing to evaluate how well the algorithm will be able to perform on newly created data. For very large data sets, several iterations are run with the training set, which may be as little as 10% of the entire set (a randomly selected 10%). Because they have few rows, smaller datasets are handled to allow for accurate results while covering the entire set; specifically, they are partitioned by the number of desired iterations. The choice of number of iterations is arbitrary, but depends on the size and properties of the dataset. One partition is set aside for testing, while the remaining partitions are used for training. The training/testing then commences for the desired number of iterations, with the partitions being rotated after each iteration (rotation meaning that the smaller testing set is rotated in and out of the larger partition "pool" that makes up the training set). The method for handling small data sets is commonly called *k*-fold cross validation, where *k* is the number of partitions [Cro06].

Steps 3 and 4 are self-explanatory. Step 5, however, requires explanation. One may wonder why, after already having reduced the problem size significantly by reducing the discernibility matrix size, it is necessary to further reduce the complexity of the problem, and thus the accuracy of the results. The answer is straightforward. When doing a full reduct search, all possibilities are examined. This means that a data set of 20 features with an average reduct size of 4 contains 16! (2.09E13) possibilities to examine. A 50 feature data set with an average reduct size of 7 has 43! (6.04E52) possibilities. One can imagine how quickly the number of possibilities, and therefore the size of the problem,

can grow. Thus, with a large number of features, even a small discernibility matrix can take a long time to search, making it necessary to restrict the features being examined.

Step 6 can take significant time to execute. It “builds” the entire discernibility matrix without actually storing it all. To be precise, it compares all rows to all other rows, but does not actually store the entire matrix. It only stores a discernibility comparison if it doesn’t contain any of the previously selected features. This method creates a highly accurate representation of the discernibility matrix, without actually taking up the potentially massive amount of memory that would otherwise be required and it only needs to be performed once. Furthermore, because all rows are examined, it is also possible to check for any additional core elements. Thus, when performing Step 7, the discernibility matrix can be further reduced in size by any new core features that have been discovered.

Step 8 finds the reducts (see Chapter 2 for the algorithm), which in general are discerns of the original information system, because discerns cannot be verified as reducts as a result of the removal of features from the full discernibility matrix. The removal of rows in the discernibility matrix containing the selected features, although greatly reducing computational time, has also removed information necessary to verify the validity of the resulting reducts. The algorithm is designed to find all reducts when given the full discernibility matrix. However, without it, it will find a mixture of discerns and unverifiable reducts.

Step 9 determines classification rules based on discerns found in Step 8. For this demonstration the simple method covered in Chapter 3 was used, since the goal was not to produce the best classifiers, but rather to quantitatively show that by reducing the search space, results can be obtained of the same quality as those obtained by the exhaustive method.

Steps 10 and 11 are straightforward. Once the rule generation is completed, the rules are applied to the test data. However, since the test data will generally be covered by more than one rule, it is necessary to implement a mechanism, by which all rules are applied to the test data, after which, the classification with the greatest number of hits "wins." In cases where the rule count is tied, a random selection of the winner class was implemented.

The preceding algorithm reduces the size of the problem space by several orders of magnitude by reducing the amount of information to be examined while still maintaining the data's integrity. In addition, the algorithm determines the accuracy of the results to allow for easy comparison to other classification methods. It will be shown that the results are of the same quality as those obtained from an exhaustive reduct search, or other methods of reduct based classification.

4.5 Results and Discussion

The following tables show results of tests on varying randomly selected portions of selected databases; three different breast cancer databases and a mushroom database. These databases were selected because they have relatively large numbers or instances and/or features compared to smaller databases like the well-known Iris database.

The databases have the following characteristics:

1) The ***wdbc*** [Wol96a] database contains 30 features, 569 instances, and 2 classes. The final decision column contains class information, specifying whether the data represents a malignant or benign diagnosis. The classifications are split 37.2% for malignant and 62.8% for benign.
2) The ***wpbc*** [Wol96b] database contains 32 features, 198 instances, and 2 classes. The class distributions are 76.2% nonrecurring and 23.8% recurring.
3) The ***bcwis*** [Wol92] database consists of 9 features, 699 instances, and 2 classes. Its class distributions are 65.5% malignant and 34.5% benign.
4) The ***mushroom*** *[Mur90]* database possesses 22 features, 8124 instances, and 2 classes. Its class distributions are 51.8% edible, and 48.2% poisonous.

The ***wdbc*** database, Table 4.2, was originally selected because it has an appropriate (with regards to computational time for both methods) number of features and instances for the purpose of testing. The next two tables, Table 4.3 and Table 4.4, depict results using the ***bcwis*** database and the ***wpbc*** database respectively. These databases were, like the ***wdbc*** database, selected for the number of instances they contained. The ***bcwis*** database, in particular, was selected because the number of its features were already reduced and pre-

scaled, while the ***wpbc*** database has more features than many of the other suitable databases available at the UCI Repository. Table 4.5 is of special interest, largely because the ***mushroom*** database is the largest dataset used in terms of instances, and is, therefore, a good indicator of time utilization. The classification results discussed in the following section for the four databases are found in Columns 3 and 6 of Tables 4.2-4.5.

The number of iterations used for each test varies from test to test due to time constraints. It simply was not practical to continue testing when individual iterations ran past an hour. Several items were tracked, among which were the averages and standard deviations for: reduct search time, number or discerns discovered, number of rules found, correct classification percentage, and total time to completion.

The following tables show the classification results, as well as other information from the test runs. The “Learn vs. Test %” column shows what percentage of the entire dataset was used for learning (finding reducts and generating rules) and what percentage was used for testing. The next three columns show results for the specific database using the exhaustive method, and the remaining three columns show the results using the statistical method presented in this chapter. With regards to the statistical method, it is important to note, that a confidence level of 99% with a confidence interval of 0.5% were chosen for use with the algorithm for all tests.

Table 4.2 – Test results for ***wdbc*** database.

	Exhaustive	Exhaustive	Exhaustive	Stat Based	Stat Based	Stat Based
Learn vs. Test %	Average # of Rules	Average Correct Classification	Avg Reduct Time (sec)	Average # of Rules	Average Correct Classification	Avg Discern Time (sec)
10/90	69	88.49	1.61	62	88.82	0.38
20/80	114	93.00	5.63	100	92.49	0.90
30/70	405	93.32	13.95	306	93.71	1.59
40/60	950	93.65	30.28	524	93.72	1.81
50/50	1706	93.80	122.23	939	94.31	2.11
60/40	2724	93.73	222.36	1217	93.55	2.64
70/30	3715	94.74	370.37	1827	93.57	3.35
80/20	4634	94.30	506.52	2394	92.76	3.87
90/10	5573	96.84	753.15	3101	95.35	4.38

Table 4.3 – Test results for ***bcwis*** database.

	Exhaustive	Exhaustive	Exhaustive	Stat Based	Stat Based	Stat Based
Learn vs. Test %	Average # of Rules	Average Correct Classification	Avg Reduct Time (sec)	Average # of Rules	Average Correct Classification	Avg Discern Time (sec)
10/90	20	86.72	0.19	14	87.86	0.13
20/80	48	92.20	0.46	39	92.34	0.29
30/70	77	93.70	0.89	45	93.47	0.52
40/60	107	93.74	1.43	83	93.83	0.82
50/50	122	94.28	2.19	96	94.58	1.12
60/40	160	94.10	3.39	110	94.40	1.46
70/30	176	95.00	4.48	130	93.91	1.80
80/20	196	94.12	5.79	148	94.56	2.18
90/10	191	94.00	6.91	143	94.89	2.46

Table 4.4 – Test results for ***wpbc*** database.

	Exhaustive	Exhaustive	Exhaustive	Stat Based	Stat Based	Stat Based
Learn vs. Test %	Average # of Rules	Average Correct Classification	Avg Reduct Time (sec)	Average # of Rules	Average Correct Classification	Avg Discern Time (sec)
10/90	3	42.64	0.66	2	41.40	0.64
20/80	18	65.92	1.98	25	68.86	0.40
30/70	50	74.46	3.59	48	74.10	0.65
40/60	60	75.59	6.57	47	75.29	1.07
50/50	80	74.14	10.96	54	75.66	1.53
60/40	102	75.76	17.45	52	72.78	1.19
70/30	166	75.68	24.40	76	72.12	1.32
80/20	266	75.50	34.70	128	73.75	1.47
90/10	394	72.75	48.58	208	78.00	1.92

Table 4.5 – Test results for ***mushroom*** database.

	Exhaustive	Exhaustive	Exhaustive	Stat Based	Stat Based	Stat Based
Learn vs. Test %	Average # of Rules	Average Correct Classification	Avg Reduct Time (sec)	Average # of Rules	Average Correct Classification	Avg Discern Time (sec)
2.5/97.5	587	96.80	5.65	164	96.89	1.23
5/95	794	98.71	24.44	234	98.30	2.83
10/90	1098	99.40	83.54	263	99.48	5.85
20/80	1327	99.82	302.31	293	99.88	14.86
30/70	1409	99.98	658.53	285	99.95	29.67
40/60	1423	99.98	1216.30	305	99.98	49.81
50/50	1427	100.00	1763.50	316	99.99	77.56
60/40	1523	100.00	2567.90	307	100.00	108.40
70/30	1550	100.00	3441.90	340	99.99	146.65
80/20	1409	100.00	4442.40	303	99.99	193.63
90/10	1159	100.00	5706.20	325	100.00	248.44

Complete results are shown in the Appendix. The average reduct/discern times were tracked because it is important to see how much time is saved by the statistical method explained here. The average total time was tracked because the time to completion shows how the statistical method impacts the other processes in the algorithm and provides a

complete picture. The numbers of rules and reducts were tracked as a way of differentiating the different tests. For example, in the 2.5/97.5 ***mushroom*** database statistical test, an average of 164 rules were discovered over 10 iterations, while for the 90/10 test an average of 325 rules were discovered. Interestingly, the average number of reducts/discerns was actually less in the 90/10 test compared to the 2.5/97.5 test, 46 vs. 98. This can be explained because the 90/10 test had several thousand more instances to use for rule generation despite having fewer discerns with which to work. The actual implications of these results and the results of other people's work on the selected databases are considered in the next section.

4.5.1 Discussion of Classification Results

The proposed hybrid statistical method gives classification results nearly as good as those achieved using other approximate methods even when using a non-optimized rule generation algorithm. The ***mushroom*** database typically permits 100% correct classification and that result has been matched as can be seen by examining the results of the **90/10** test for which the discrepancy between the statistical method and the exhaustive method is 0.00%. For comparison, using the SolarC [Sta02] (classification) algorithm achieved a result of 99.99% correct classification in 877.98 seconds. This SolarC test when combined with the subsequent SolarC results shows that the SolarC algorithm goes a good job classifying smaller databases. However, once databases approach the size of the ***mushroom*** database a large increase in computational time

occurs. The SolarC test took roughly 3.5 times longer than the present statistical method, showing that despite its classification performance it does not scale well on computational time, thus limiting its usefulness for larger databases.

For the ***wdbc*** database using the **90/10** test, the difference in classification accuracy between the exhaustive (96.84%) and statistical (95.35%) based test is 1.49%. For this database, the exhaustive method almost universally performs better than the statistical one. However, considering the savings in computational time, the difference in the quality of results can be thought of as negligible, and it is a given that a more sophisticated rule generation algorithm would likely close the gap further. The SolarC algorithm yielded a result of 94.64% correct classification in 113.34 seconds. In their work, Chen et al. [Che05] presented results of using several classification methods including neural network, reduct, and others in an ensemble classifier. The ensemble classifier performed the best and was able to get as high as 96.14% correct classification of malignant cancers and 95.42% correct classification of benign cancers. The best predictive accuracy on this database according to the information in the UCI repository [Wol96a] is 97.50%, using one separating plane in the 3-D space of Worst Area, Worst Smoothness and Mean Texture, obtained by applying repeated 10-fold cross validations. These results show that the hybrid statistical method's results are representative of other methods.

There are many reported classification results for the ***bcwis*** database, likely because it is preprocessed, meaning there are no variations in results as a result of differences in preprocessing among the various tests. Using the developed method and looking at the **90/10** (***bcwis***) test, the difference in classification accuracy between the exhaustive (94.00%) and statistical (94.89%) based test is 0.89% in favor of the statistical method. In contrast, the **70/30** test shows a difference of 1.09% in favor of the exhaustive test. Note that the results using the exhaustive method are generally better as they are for the ***wdbc*** database. However, the results for the statistical test are better for the ***bcwis*** database, which is something of an anomaly, although the scale of the difference is not highly significant.

The ***bcwis*** database was also tested using the SolarC algorithm, and received a 95.65% correct classification result in 25.125 seconds. As before, other testing methods produce both better and worse results, such as those found in [Hon04], which is based in part on fuzzy sets; BCFS-1: 96.00%, BCFS-1,2: 96.75%, SVM: 97.33%, FNN-SWEEP: 93.00%. Additional results for comparison can be found in [Soo97, Kol04a, Kol04b]. Leifler also made some use of the breast cancer databases in her thesis [Lei02], however, only a relatively small subset of 286 elements was used. Her work is relevant because she tested a dynamic reduct classification approach against another method. Unfortunately, due to either the size or composition of the data used, correct classification results only reached 93.64%. These examples illustrate that the results can vary either way depending on the individual database, the size of the learning set and the algorithm.

The ***wpbc*** database is a less commonly used database, but was chosen because it possessed the most features of the databases that have been examined. As before, examine the **90/10** (***wpbc***) test; the difference in classification accuracy between the exhaustive (72.75%) and statistical (78.00%) based test is 5.25% in favor of the statistical method. The results for this database tend to flip-flop from one side to the other, although, typically, the difference isn't as extreme as what is observed in the **90/10** test.

One test [Ben92] using the ***wpbc*** database gave a correct classification result of 86.3%. As with the other databases, the ***wpbc*** database was also tested using the SolarC algorithm, which was able to get as high as 89.47% correct classification in 96.47 seconds. These results illustrate that there is room for improvement in the rule generation algorithm used in this paper. However, to reiterate, the main goal is not to produce the best possible overall algorithm, but to show the utility of the statistically found discerns. If there were a greater number of data points (rows) to work from, it is not inconceivable to assume that the correct classification percentage would be notably greater. However, with the limited number of data points the ***wpbc*** database is simply too "individual" to be easily classified. This caused the rules that the rule generation algorithm produced to be too specific and, therefore, caused many rules to be dropped, causing a decrease in the correct classification percentage. Other factors may also be present, such as overly noisy or contradictory data; however, it would require a more in depth analysis of the data set to verify this.

4.5.2 Computational Time

Figure 4.3 graphically compares the time to completion using either the Exhaustive or the Statistical methods for reduct determination for the ***wdbc*** database. Results in seconds are plotted vs. the percent of the full dataset used for training. Notice how much faster the computational time increases (approximately exponential) for the exhaustive method compared to the almost linear progression of the statistical method.

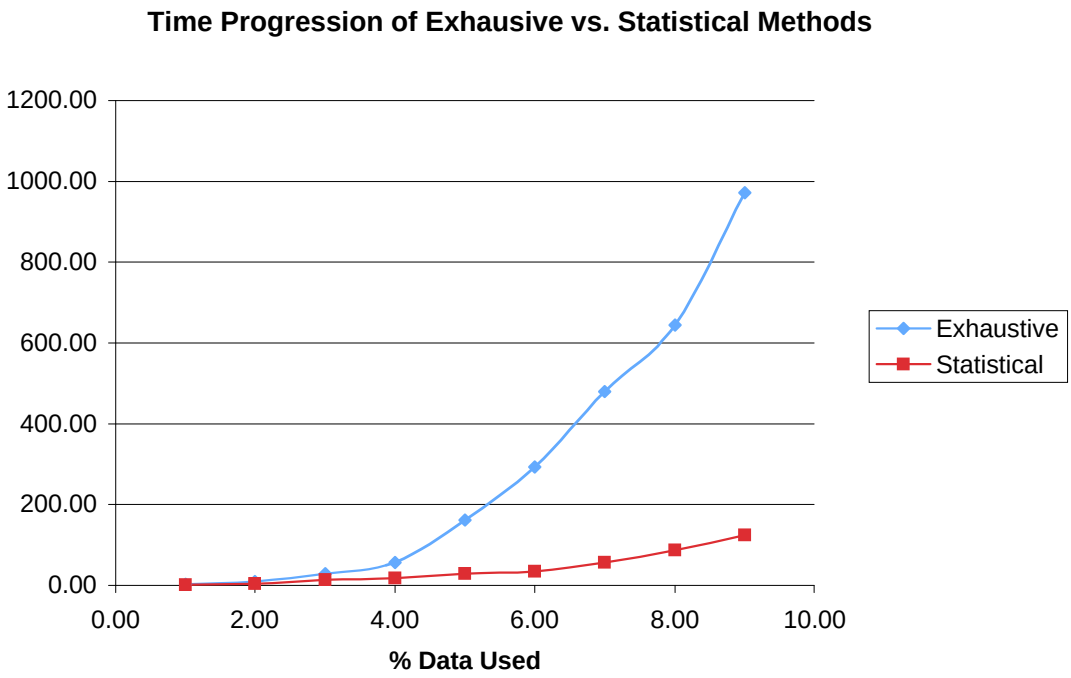

Figure 4.3 – Exhaustive vs. statistical time to completion for ***wdbc*** database.

Also, note, in regards to Tables 4.2-4.5, that the statistical method generates both fewer reducts and fewer rules. Fewer reducts are a result of the reduced size of the discernibility

matrix, while the fewer rules can be traced to the reduction in the number of reducts (discerns) discovered. And yet, despite the reduction in information, the statistical method produces results of nearly equivalent quality, meaning that the results are "good enough" considering the massive time saving potential of the algorithm. (The level of "good enough" is determined at the outset when selecting the desired confidence interval level.)

As with the Figure 4.3 above, Figure 4.4 shows the improvement in computation time of the statistical method compared to the exhaustive method (linear vs. exponential growth in computational time vs. % of the data set used) for the ***mushroom*** database.

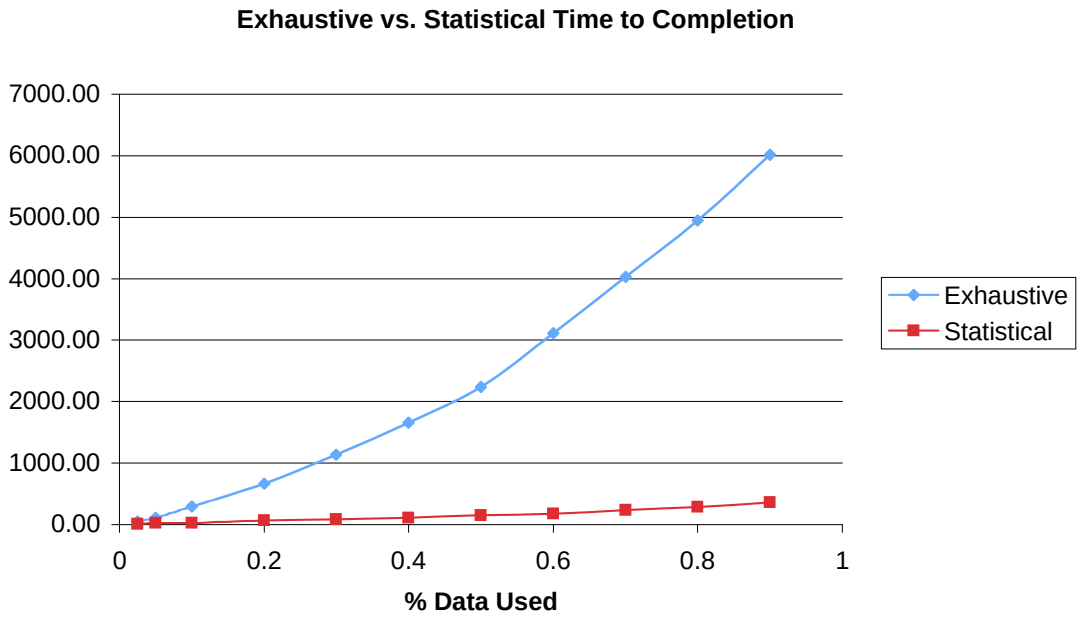

Figure 4.4 – Exhaustive vs. statistical time to completion for ***mushroom*** database.

In Figure 4.4, using a 90% training set of the mushroom database, the hybrid statistical method takes roughly 350 seconds, while the exhaustive method takes well over an hour to complete a single iteration of the test. Also, note that it appears that the curves in Figure 4.3 have a greater slope than the curves in Figure 4.4. This is almost certainly related to the higher number of features present in the ***wdbc*** database. This helps explain the reason for Step 5 in the algorithm; the presence of a greater number of features causes the time needed for reduct generation to grow exponentially, thus necessitating the inclusion of a cut-off for the number of features allowed in the algorithm as well as a method for selecting the features utilized.

4.5.3 Confidence Level and Confidence Interval Values

Up to this point, the values of the Confidence Interval and Confidence Interval have been largely ignored. All the tests previously mentioned have held these two variables to a constant value of 99% for the Confidence Level and 0.5% for the Confidence Interval. In this section the effect on test results of varying the values of these variables is examined. Table 4.6 provides the results for several different combinations of Confidence Level and Confidence Interval. Each test was performed on the ***wdbc*** database for 20 iterations. By changing the Confidence Interval and adjusting the Confidence Level accordingly, the number of samples needed to reach the desired Confidence Values changes correspondingly. While greater Confidence Intervals require fewer samples, they produce

less accurate results. In fact, by the time the 78% and 80% Confidence Levels are reached, the population size will only amount to a few samples.

Table 4.6 – Classification results for varying confidence values.

Confidence Level	Confidence Interval	Avg. Discern Time (s)	Average Num Discerns	Average Total Time (s)	Average Correct Classification
99.90%	0.10%	10.73	933	130.38	93.07%
99.75%	0.25%	8.82	987	122.64	95.35%
99.50%	0.50%	5.59	897	126.93	93.77%
99.00%	1.00%	3.29	1022	129.75	95.61%
98.00%	2.00%	2.61	1226	101.40	94.56%
97.00%	3.00%	2.52	1301	132.10	93.77%
96.00%	4.00%	2.47	1317	146.34	95.00%
95.00%	5.00%	2.43	1287	175.74	93.77%
94.00%	6.00%	2.46	1297	203.75	94.65%
93.00%	7.00%	2.50	1431	177.45	95.35%
90.00%	10.00%	8.62	6275	175.70	95.96%
80.00%	20.00%	16.27	8063	249.78	93.77%
70.00%	30.00%	20.99	8473	228.72	94.74%

Surprisingly, the statistical variable values seem to have relatively little effect on the resulting average correct classification percentage values. For examples, the 90% Confidence Level test actually produced better average results than any of the other tests. In fact, the only apparent advantage of using higher confidence values is their effect toward reducing the algorithm's computations time. Note, that as the confidence level declines and the interval widens, the average number of discerns increases.

Figure 4.5, below, illustrates how the confidence interval affects the number of discerns generated by the algorithm. Both the ***wdbc*** and ***bcwis*** databases were examined for this test. At the 90% Confidence Level, with its corresponding 10% Confidence Interval, there was a sharp increase in the number of discerns discovered by the algorithm. The reason for this is that a threshold in the sampling quality was reached. The sampled discernibility matrix differed enough from the true discernibility matrix that when the discern generation matrix was created it ended up being much larger than expected, thus, the reason for both the greater number of discerns and the corresponding increase in time. This threshold can be observed in Figure 4.5 below in the 7%-10% range.

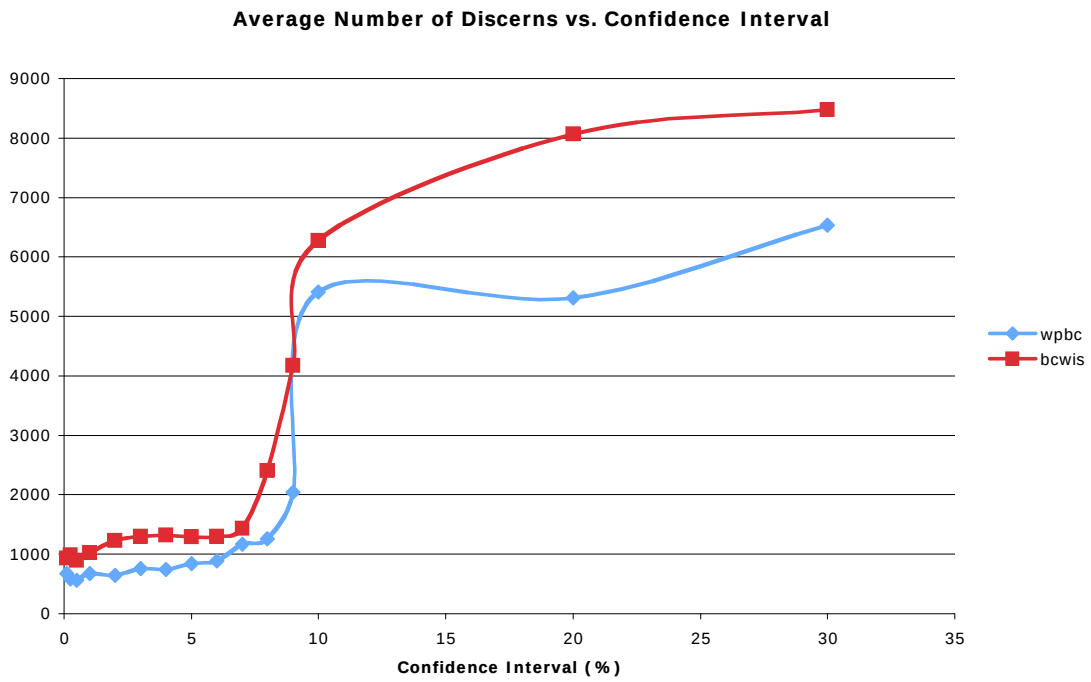

Figure 4.5 – Average number of discerns vs. confidence interval.

The actual position of the level jump will vary from database to database, however it is expected to be present in varying extremes in all databases. By observing Figure 4.5 and Table 4.6 it is apparent that the main effect of the Confidence Values is their impact on time. In all experiments the sampling quality threshold was always to the right of the confidence interval size that corresponds to the minimum total discern generation time. From this analysis it is possible to realize some additional time improvements by adjusting the Confidence Values so that the algorithm takes less time in the sampling stage without sacrificing too much accuracy and causing the resulting discernibility matrix to be too large. For the ***wdbc*** database, this point would be around the 98% confidence level with a 2% confidence interval, since that region is where the time taken is at its lowest as seen in Table 4.6. Unfortunately, there was no discernible pattern to the classification results with regards to the Confidence Values.

4.6 Comparison of Standard and Hybrid Statistical Methods

This section provides some results using the standard statistical method mentioned in Section 4.3, and compares them to the results achieved using the hybrid method that has been the focus of this chapter. Table 4.7, shows the results of the standard statistical method applied to the ***mushroom*** database. If the results of Table 4.7 are compared to the results shown in Table 4.5 (also see Table A4), it becomes apparent, especially from looking at the reduct generation time, that the standard statistical method easily beats the hybrid statistical method in computation time. On the other hand, while it is not highly

obvious when using the ***mushroom*** database, it is still apparent that the classification results are significantly less accurate. This is especially apparent as both exhaustive and hybrid methods produced several instances of 100% correct classification, while the standard statistical method produced none.

Table 4.7 – Standard method test results for ***mushroom*** database.

Learn/Test	Avg. Discern Time (sec.)	Average Num Discerns	Average # of Rules	Average Correct % Classification	Average Total Time (sec)
0.025	1.02	81	148	96.46	7.51
0.05	2.31	77	207	98.56	14.23
0.1	3.68	79	289	99.48	22.50
0.2	4.38	100	412	99.73	48.27
0.3	4.69	101	492	99.89	71.23
0.4	4.83	96	516	99.92	82.68
0.5	4.82	105	548	99.89	101.76
0.6	5.01	111	596	99.92	124.92
0.7	5.06	122	752	99.95	179.69
0.8	5.20	131	782	99.92	208.34
0.9	5.24	98	652	99.93	139.10

The utility of the standard statistical method can be seen from its ability to handle datasets that would be prohibitive in size for the hybrid method because of its reliance on the one time calculation of the reduced full discernibility matrix. In fact, if the reduct calculation times presented in Table 4.7 and Table A4 are examined, it becomes apparent that not only does the standard method drastically reduce Discern computation time compared to the hybrid method, but that the computation time for the standard method actually levels out at around five seconds. For a comparison of the standard vs. hybrid

discern generation times see Figure 4.6. The leveling occurs because of the S_{size} limit mentioned in Section 4.2 and the discernibility matrix size threshold for feature removal (the point at which features stop being removed from the sampled discernibility matrix). These results verify the expected uses for the two versions of the statistical method.

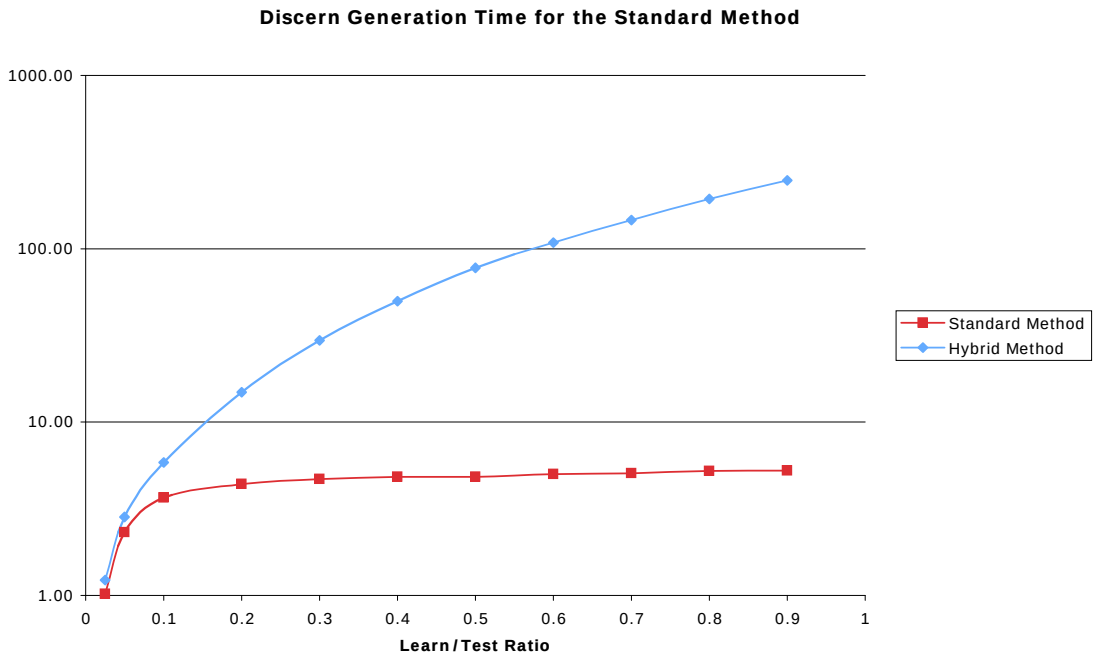

Figure 4.6 – Discern generation time for the standard method.

The hybrid version works best with medium sized databases and in cases where accuracy is of high importance. The standard version of the method will work best for extremely large databases (databases with tens of thousands of entries), where achieving results in a timely manner is more important than a maximum accuracy in specifying the classification rules. Correct classification is a function of both – the classifier accuracy

and data separability. Thus, if data is not statistically separable, it would not make sense to strive for the maximum precision in defining the classifier, since small differences in the classifier performance may be overshadowed by large classification error resulting from data inseparability.

4.7 Conclusions

The methods introduced in this chapter, while lacking the completeness of the exhaustive methodology from Chapter 2 to which they are compared, are shown to generate results at a level of quality on par with other methods while saving massively in computational time. The statistical method used an optimized exhaustive algorithm, integrating the statistical reduction methods into the procedure, to create an algorithm for which the computational time is nearly linear vs. the amount of data examined. A more sophisticated rule generation algorithm may improve the classification results for the methods examined; however, increased sophistication means both greater complexity and increased computational time.

The quality of the results is tied at least in part to the way features are selected to reduce computational time. To clarify, the selected features are the most common differentiating elements in each database, meaning that in most circumstances their presence in a reduct/discern results in a shorter reduct/discern. These features are then present in every single discern generated by the algorithm. Because of the feature's common nature, they

tend to be good classifiers. However, their presence also eliminates a greater portion of the available discern pool. And while much of the “pool” is redundant and unnecessary, the missing portions are likely the cause of the slight decrease in overall correct classification observed in the statistical tests. The size of the database being tested can also affect the quality of the results. As was mentioned with regards to the ***wpbc*** database, too small a database, or one whose data elements are too “unique”, will result in rules that are too precise, thus causing them to be dropped from consideration.

5. Thesis Conclusions

5.1 Conclusions

This thesis has examined reduct based classification with the hope of improving on one of the greatest drawbacks of such work, the computational time. In Chapter 1, the background of reduct generation and research in this area was examined. Several other peoples' works were referenced as useful source material, and other methods for finding reducts have been described.

In the second chapter, the basic algorithm for finding reducts was examined for potential ways to accelerate performance. The basic notation and underlying math were also briefly covered. Several potentially useful algorithm modifications were found, coded, and examined by testing for results against a well-known program that was also capable of finding reducts exhaustively. Results were promising and showed substantial time improvements against the mentioned comparison program. However, also noted, was the remaining NP-complete computational nature of the problem, and that computational time was still too great for the bulk of databases. This was illustrated by a dataset test of a 40 features by 500 instances whose time to completion exceeded 16 hours.

Chapter 3 provided background information on how to produce classification results using reducts/discerns. Since reducts by themselves cannot be used to classify data, it was necessary to provide a way for them to be used to generate meaningful results. In order to do this, Chapter 3 introduced rule generation via reducts for classification purposes, and provided the necessary background before proceeding to Chapter 4.

At the end of Chapter 2, it was mentioned that the use of discerns, as opposed to reducts, is a promising approach to improving classification performance. Results in Chapter 4 support this assertion. Chapter 4 showed, that by reducing computational time further and generating discerns as opposed to reducts, it is possible to produce results approaching or of the same quality as other methods, including the exhaustive method of Chapter 2. The reason for this similarity in quality arises because the majority of those reducts found using an exhaustive algorithm (there are often many thousands for some databases) are redundant and unneeded. They may potentially provide some improvement in the end classification results. However, the chore of producing them, and then determining which ones produce the best rules is simply too time consuming. On the other hand, the hybrid discern based method introduced in Chapter 4 produces fewer discerns at a much greater rate, and, as is true of the exhaustive method, doesn't use all of them. While the hybrid discern based method is a significant improvement over the exhaustive version, it still has some issues with exceptionally large databases. To that end, Section 4.6 discusses the standard statistical method, which while less accurate, can produce reducts at a much better rate (reduced computation time).

This thesis has delved into the core of reduct generation in an attempt to find ways of decreasing the computational time while maintaining good results. Initially, the basic reduct generation algorithm was examined and ways to improve its speed were sought. When this was verified to be insufficient, research continued into other slightly less precise methods for generating reduct based classifiers. Eventually, two methods based on statistical sampling were examined, with the more accurate hybrid method being chosen for the focus of the research. Results show that for the databases tested, the hybrid method performed well in terms of both accuracy and time.

5.2 Future Work

There remains significant room for improvement in the algorithm proposed in Chapter 4. Many such potential improvements have been implemented in other work. For example, the discretization step (labeling) and the feature selection portion of the algorithm could be optimized using entropy measures in labels. Likewise, the capacity to handle textual input could be added, or the ability to recognize and handle patterns in the input data could be utilized. The statistical feature selection method could potentially be improved as well. For example, other factors for feature selection, aside from simple representation within a sampled discernibility matrix, could be introduced. However, such changes would most likely be at the expense of computational time. There is also potential for additional research into the standard statistical method (see section 4.6), since it is clearly

superior in terms of time usage to any of the other methods examined. Such research might focus on improving the classification accuracy without too significantly affecting the computation time.

Also, a relatively simple rule generation algorithm was used in this thesis. There exist varieties of other algorithms for rule generation, which might improve the quality of the results, but it is expected that they might increase the computational time. One such algorithm, for example, is the a priori based method touched upon in Section 4.6. There is little doubt, that an improved rule generation method would improve the correct classification percentages produced by any of the algorithm presented herein. However, sorting this out is left for future work.

References

[Agr94] R. Agrawal and S. Srikant, “Fast Algorithms for Mining Association Rules in Large Databases,” in *Proc. International Conference on Very Large Data Bases*, 1994, pp. 487–499.

[Baz94] J.G. Bazan, A. Skowron, and P. Synak, “Dynamic reducts as a tool for extracting laws from decision tables”, in *Proc. International Symposium on Methodologies for Intelligent Systems*, vol. 869 of Lecture Notes in Artificial Intelligence, 1994, pp. 346–355.

[Baz98] J. G. Bazan, “A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision tables”, in *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, L. Polkowski and A. Skowron, Eds. Berlin, Germany: Springer-Verlag, 1998, pp. 322-365.

[Ben92] K. P. Bennett, "Decision Tree Construction Via Linear Programming," in *Proc of the 4th Midwest Artificial Intelligence and Cognitive Science Society*, 1992, pp. 97-101.

[Bjo97] A.T. Bjorvand, J. Komorowski, “Practical Applications of Genetic Algorithms for Efficient Reduct Computation,” *Wissenschaft & Technik Verlag*, vol. 4, 1997, pp. 601-606.

[Bor03] C. Borgelt, “Efficient Implementations of Apriori and Eclat,” in *Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, Melbourne, FL, November 19, 2003.

[Che05] Y. Chen, A. Abraham, and B. Yang, “Hybrid Neurocomputing for Breast Cancer Detection,” in *The Fourth International Workshop on Soft Computing as Transdisciplinary Science and Technology*, 2005, pp.884-892.

[Cro06] *Cross Validation* (2006, June 21 – last update). [Online]. Available: http://en.wikipedia.org/wiki/Cross-validation [2006, September 15]

[Dub90] D. Dubois, "Rough fuzzy sets and fuzzy rough sets," *International Journal of General Systems*, vol. 17, 1990, pp. 191–209.

[Fis88] R. A. Fisher, Iris Plants Database, UCI Machine Learning Repository, University of California, Department of Information and Computer Science, Irvine, California, 1988. (http://www.ics.uci.edu/~mlearn/databases/iris/iris.data)

[Gam99] A. Gammerman and V. Vovk., “Kolmogorov Complexity: Sources, Theory and Applications”, *The Computer Journal*, vol. 42, 1999, pp. 252-255.

[Her05] J. Herbert and J.T. Yao, “Time-Series Data Analysis with Rough Sets,” in *Proc. 4th International Conference on Computational Intelligence in Economics and Finance* (CIEF), Salt Lake City, July 21-26, 2005, pp. 908-911.

[Hon04] H. Takahashi and H. Honda, “New cancer diagnosis method on the basis of fuzzy theory and boosting,” Unpublished, Nagoya University, 2004.

[Hu03] K. Hu, Y. Lu, and C. Shi, “Feature Ranking in Rough Set,” *AI Communications*, vol. 16, 2003, pp. 41-50.

[Im05] S. Im, Z. W. Ras, A. Dardzinska, "Building a security-aware query answering system based on hierarchical data masking", in *Computational Intelligence in Data Mining*, Nova Scotia: Saint Mary's Univ., 2005, pp. 55-62.

[Jon74] D. S. Johnson. “Approximation algorithms for combinatorial problems”, *Journal of Computer and System Sciences*, vol. 9, 1974, pp. 256–278.

[Kol04a] P. Mylonas, M Wallace, and S Kollias, “Using k-nearest neighbor and feature selections as an improvement to hierarchical clustering,” presented at 3rd Hellenic Conf. on Artificial Intelligence, Samos, Greece, 2004.

[Kol04b**]** M. Wallace, N. Tsapatsoulis, and S. Kollias, “Intelligent initialization of resource allocating RBF networks,” *Neural Networks*, vol. 18, March 2005, pp. 117-122.

[Lei02] O. Leifler, *Comparison of LEM2 and a Dynamic Reduct Classification Algorithm*, MS Thesis, Linköpings University, 2002, [Online]. Available: http://www.diva-portal.org/diva/getDocument?urn_nbn_se_liu_diva-1856-1__fulltext.pdf [February 2, 2007]

[Mel96] M. Melanie, *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

[Mos05] M. Moshkov and M. Piliszczuk, “On construction of partial reducts and bounds on their complexity,” in *Systemy wspomagania decyzji*, Institute of Computer Science US, Poland, Katowice, 2005, pp. 102-106.

[Mos07] M. Moshkov, M. Piliszczuk, and B. Zielosko, “On Construction of Partial Reducts and Irreducible Partial Decision Rule,” *Fundamenta Informatica, vol. 75(1-4), 2007, pp. 357-374.*

[Mur90] Patrick M. Murphy, David. W. Aha, Mushroom Database, UCI Machine Learning Repository, University of California, Department of Information and Computer Science, Irvine, California, 1987, [Online].
Available: http://www.ics.uci.edu/~mlearn/databases/mushroom/agaricus-lepiota.data [May 23, 2006]

[Nel01] D.E. Nelson, *High Range Resolution Radar Target Classification: A Rough Set Approach*, PhD Thesis, Ohio University, 2001, [Online]. Available: http://www.ent.ohiou.edu/~starzyk/network/Research/Thesis/Dale_Nelson_dissertation.pdf [May 4, 2006]

[Ohr99] A. Ohrn, *Discernibility and Rough Sets in Medicine: Tools and Applications,* PhD Thesis, Norwegian University of Science and Technology, 1999, [Online]. Available: http://www.idi.ntnu.no/~aleks/thesis/ [March 28, 2006]

[Paw82] Z. Pawlak, "Rough Sets," *International Journal of Computer and Information Sciences*, vol. 11, 1982, pp. 341-356.

[Paw84] Z. Pawlak, "Rough Sets and Classification", *International Journal of Man-Machine Studies*, vol. 20(5), 1984, pp. 469-483.

[Paw85] Z. Pawlak, "Rough Sets and Fuzzy Sets", *International Journal of Man-Machine Studies*, vol. 21(2), 1985, pp. 99-102.

[Paw91] Z. Pawlak, "Rough Sets - Theoretical Aspects of Reasoning About Data," in *Kluwer Academic Publ*, 1991.

[Pol06] R. Polikar, “Ensemble based systems in decision making,” in *IEEE Circuits and Systems*, vol. 2(3), 2006, pp. 21-45.

[Pyl99] D. Pyle, *Data Preparation for Data Mining*. San Francisco: Morgan Kaufman, 1999.

[Ras06] Z. W. Ras and A. Dardzinska , "Extracting Rules from Incomplete Decision Systems: System ERID", in *Studies in Computational Intelligence Vol. 9: Foundations and Novel Approaches in Data Mining*, T.Y. Lin, S. Ohsuga, C.J. Liau, and X. Hu, Eds. Springer, 2006, pp. 143-154.

[Ras07] Z. W. Ras and S. Im, "Data Confidentiality and Chase-Based Knowledge Discovery", in *Encyclopedia of Data Warehousing and Mining,* 2nd ed., J. Wang, Ed., Idea Group Inc, 2007, will appear.

[RSES2] RSES2.2 User’s Guide, Warsaw University, 2005, [Online]. Available: http://logic.mimuw.edu.pl/~rses/ [November 29, 2005]

[Slo92] A. Skowron and C. Rauszer, “The Discernibility Matrices and Functions in Information Systems,” in *Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory*, Dordrecht, The Netherlands: Kluwer, 1992, pp. 331-362.

[Soo97] T. Lin and V. Soo, “Pruning Fuzzy ARTMAP Using Minimum Description Length Principle in Learning from Clinical Databases,” in *Proc. of the 9th International Conference on Tools with Artificial Intelligence,* 1997, pp. 396-403.

[Sta99] J. A. Starzyk, D. E. Nelson, and K. Sturtz, “Reduct Generation in Information Systems,” in *Bulletin of International Rough Set Society*, vol. 3(1/2), 1999.

[Sta00] J. A. Starzyk, D. E. Nelson, and K. Sturtz, “A Mathematical Foundation for Improved Reduct Generation in Information Systems,” in *Journal of Knowledge and Information Systems*, vol. 2(2), 2000, pp.131-146.

[Sta02] J. Starzyk and Z. Zhu, "Software Simulation of a Self-Organizing Learning Array System", presented at The 6th IASTED Int. Conf. Artificial Intelligence & Soft Comp. (ASC 2002), Banff, Alberta, Canada, July 17-19, 2002.

[Tse05] T. Tseng, Y. Kwon, and Y. Ertekin, “Feature-based rule induction in machining operation using rough set theory for quality assurance,” in *Robotics and Computer-Integrated Manufacturing*, vol. 21(6), 2005, pp. 559-567.

[Vin99] S. Vinterbo and A. Øhrn, “Minimal approximate hitting sets and rule templates, in Predictive Models in Medicine: Some Methods for Construction and Adaptation,” Department of Computer and Information Science, NTNU report 1999:130, 1999.

[Wol92] W.H Wolberg, O. L. Mangasarian, Wisconsin Breast Cancer Database (bcwis), UCI Machine Learning Repository , University of California, Department of Information and Computer Science, Irvine, California, 1991, [Online]. Available: http://www.ics.uci.edu/~mlearn/databases/breast-cancer-wisconsin/ breast-cancer-wisconsin.data [February 10, 2006]

[Wol96a] W.H Wolberg, W. N. Street, O. L. Mangasarian, Wisconsin Diagnostic Breast Cancer (WDBC) Database, UCI Machine Learning Repository, University of California, Department of Information and Computer Science, Irvine, California, 1995, [Online]. Available: http://www.ics.uci.edu/~mlearn/databases/breast-cancer-wisconsin/wdbc.data [February 10, 2006]

[Wol96b] W.H Wolberg, W. N. Street, O. L. Mangasarian, Wisconsin Prognostic Breast Cancer (WPBC) Database, UCI Machine Learning Repository, University of California, Department of Information and Computer Science, Irvine, California, 1995, [Online]. Available: http://www.ics.uci.edu/~mlearn/databases/breast-cancer-wisconsin/wpbc.data [February 10, 2006]

[Wro95] Jakub Wroblewski, “Finding minimal reducts using genetic algorithms (extended version).”, in *Proc. Second International Joint Conference on Information Sciences*, 1995, pp. 186–189.

Appendix – Full Tables

The tables in this Appendix contain the full set of information pertaining to the database results portrayed in Section 4.5, and are placed here to allow for a more thorough examination of the tests.

Table A1 – Full test results for the ***wdbc*** database.

Method	Learn vs. Test %	Iterations	Avg. Reduct Time (s)	StdDev Rtime	Average # of Rules	StdDev # of Rules	% Average Correct Classification	Std Dev Correct	Average Total Time	Std Dev Time (s)
Exhaustive	10/90	20	1.61	0.32	69	14.00	88.49	5.36	2.91	0.45
Exhaustive	20/80	20	5.63	0.78	114	21.00	93.00	1.55	10.02	1.84
Exhaustive	30/70	20	13.95	1.80	405	68.00	93.32	0.86	28.69	3.71
Exhaustive	40/60	10	30.28	2.44	950	124	93.65	1.22	57.14	6.79
Exhaustive	50/50	10	122.23	20.13	1706	158	93.80	1.22	162.08	23.15
Exhaustive	60/40	10	222.36	16.76	2724	338	93.73	1.65	293.05	30.64
Exhaustive	70/30	20	370.37	31.04	3715	277	94.74	1.47	479.65	37.98
Exhaustive	80/20	10	506.52	21.58	4634	124	94.30	1.19	644.00	34.24
Exhaustive	90/10	5	753.15	34.66	5573	542	96.84	3.14	972.30	91.93
Stat Based	10/90	20	0.38	0.06	62	11	88.82	5.11	1.40	0.24
Stat Based	20/80	20	0.90	0.17	100	15	92.49	1.48	3.86	1.56
Stat Based	30/70	20	1.59	0.20	306	72	93.71	0.92	14.46	4.11
Stat Based	40/60	20	1.81	0.35	524	167	93.72	1.44	18.41	6.49
Stat Based	50/50	20	2.11	0.22	939	302	94.31	1.29	29.10	11.71
Stat Based	60/40	20	2.64	0.17	1217	383	93.55	1.27	35.13	16.13
Stat Based	70/30	20	3.35	0.31	1827	476	93.57	2.17	56.99	26.20
Stat Based	80/20	20	3.87	0.39	2394	700	92.76	1.89	86.99	38.65
Stat Based	90/10	20	4.38	0.45	3101	692	95.35	2.63	125.06	35.67

Table A2 – Full test results for the ***bcwis*** database.

Method	Learn vs. Test %	Iterations	Avg. Reduct Time (s)	StdDev Rtime	Average # of Rules	StdDev # of Rules	% Average Correct Classification	Std Dev Correct	Average Total Time	Std Dev Time (s)
Exhaustive	10/90	20	0.19	0.01	20	5	86.72	5.12	0.79	0.15
Exhaustive	20/80	20	0.46	0.01	48	8	92.20	1.76	1.58	0.22
Exhaustive	30/70	20	0.89	0.03	77	11	93.70	1.25	2.73	0.38
Exhaustive	40/60	20	1.43	0.02	107	15	93.74	1.19	3.73	0.44
Exhaustive	50/50	20	2.19	0.05	122	18	94.28	1.10	4.88	0.74
Exhaustive	60/40	20	3.39	0.11	160	20	94.10	1.18	7.01	0.96
Exhaustive	70/30	20	4.48	0.19	176	25	95.00	1.84	8.35	1.28
Exhaustive	80/20	20	5.79	0.17	196	30	94.12	2.29	10.14	1.63
Exhaustive	90/10	20	6.91	0.06	191	11	94.00	3.59	10.43	0.52
Stat Based	10/90	20	0.13	0.01	14	4	87.86	3.58	0.45	0.08
Stat Based	20/80	20	0.29	0.01	39	5	92.34	1.41	0.94	0.16
Stat Based	30/70	20	0.52	0.01	45	7	93.47	1.14	1.49	0.26
Stat Based	40/60	20	0.82	0.05	83	12	93.83	1.32	1.96	0.39
Stat Based	50/50	20	1.12	0.03	96	18	94.58	1.51	2.54	0.52
Stat Based	60/40	20	1.46	0.04	110	22	94.40	1.46	3.18	0.64
Stat Based	70/30	20	1.80	0.04	130	27	93.91	1.65	3.71	0.71
Stat Based	80/20	20	2.18	0.09	148	28	94.56	2.61	4.64	1.01
Stat Based	90/10	20	2.46	0.04	143	27	94.89	3.02	4.67	0.96

Table A3 – Full test results for the ***wpbc*** database

Method	Learn vs. Test %	Iterations	Avg. Reduct Time (s)	StdDev Rtime	Average # of Rules	StdDev # of Rules	% Average Correct Classification	Std Dev Correct	Average Total Time	Std Dev Time (s)
Exhaustive	10/90	20	0.66	0.14	3	6	42.64	11.93	1.15	0.18
Exhaustive	20/80	20	1.98	0.24	18	8	65.92	8.17	3.09	0.24
Exhaustive	30/70	20	3.59	0.60	50	20	74.46	1.35	5.41	0.68
Exhaustive	40/60	20	6.57	0.89	60	16	75.59	2.59	9.74	1.39
Exhaustive	50/50	20	10.96	1.42	80	29	74.14	2.71	16.98	2.47
Exhaustive	60/40	20	17.45	1.75	102	31	75.76	4.15	25.62	2.28
Exhaustive	70/30	20	24.40	1.83	166	38	75.68	4.90	36.65	3.43
Exhaustive	80/20	20	34.70	2.97	266	41	75.50	6.86	52.14	4.56
Exhaustive	90/10	20	48.58	1.83	394	55	72.75	6.38	71.09	5.12
Stat Based	10/90	20	0.64	0.13	2	2	41.40	11.99	1.19	0.19
Stat Based	20/80	20	0.40	0.06	25	10	68.86	4.75	1.32	0.24
Stat Based	30/70	20	0.65	0.13	48	19	74.10	2.06	2.02	0.33
Stat Based	40/60	20	1.07	0.20	47	12	75.29	1.43	3.06	0.44
Stat Based	50/50	20	1.53	0.26	54	20	75.66	2.74	4.97	1.22
Stat Based	60/40	20	1.19	0.64	52	24	72.78	4.55	5.29	2.44
Stat Based	70/30	20	1.32	0.77	76	33	72.12	4.28	7.37	3.36
Stat Based	80/20	20	1.47	0.70	128	71	73.75	5.65	11.57	5.48
Stat Based	90/10	20	1.92	1.19	208	88	78.00	7.15	16.26	6.48

Table A4 – Full test results for the ***mushroom*** database

Method	Learn vs. Test	Itera-tions	Avg. Reduct time	StdDev. Rtime	Average # of Reducts	StdDev. #Red	Average # of rules	StdDev # Rules	Average Correct Classification	Std Dev class	Average total time	Std Dev Time
Exhaustive	2.5/97.5	10	5.65	0.07	592	189	587	105	96.80	1.62	50.34	24.33
Exhaustive	5/95	10	24.44	3.60	479	133	794	132	98.71	0.69	109.09	40.94
Exhaustive	10/90	10	83.54	9.88	394	87	1098	221	99.40	0.24	291.34	97.51
Exhaustive	20/80	5	302.31	16.59	316	25	1327	135	99.82	0.22	662.18	111.39
Exhaustive	30/70	5	658.53	45.73	275	37	1409	97	99.98	0.03	1134.50	63.20
Exhaustive	40/60	5	1216.30	46.30	272	32	1423	161	99.98	0.06	1659.50	207.38
Exhaustive	50/50	5	1763.50	45.91	222	17	1427	109	100.00	0.00	2240.80	111.39
Exhaustive	60/40	5	2567.90	40.20	211	5	1523	115	100.00	0.00	3113.40	123.15
Exhaustive	70/30	5	3441.90	25.80	209	5	1550	62	100.00	0.00	4026.60	41.35
Exhaustive	80/20	5	4442.40	15.15	203	0	1409	90	100.00	0.00	4947.00	82.51
Exhaustive	90/10	5	5706.20	88.35	203	0	1159	283	100.00	0.00	6016.30	165.05
Stat Based	2.5/97.5	10	1.23	0.09	98	49	164	45	96.89	0.72	10.18	5.85
Stat Based	5/95	10	2.83	0.23	132	100	234	88	98.30	0.94	24.40	14.91
Stat Based	10/90	10	5.85	0.24	75	57	263	92	99.48	0.29	28.84	15.58
Stat Based	20/80	10	14.86	0.31	61	42	293	107	99.88	0.07	68.91	53.62
Stat Based	30/70	10	29.67	0.53	48	26	285	93	99.95	0.06	83.83	38.26
Stat Based	40/60	10	49.81	0.77	47	16	305	42	99.98	0.04	112.88	23.56
Stat Based	50/50	10	77.56	0.09	51	29	316	84	99.99	0.03	149.30	35.68
Stat Based	60/40	10	108.40	1.60	40	12	307	64	100.00	0.01	177.15	0.60
Stat Based	70/30	10	146.65	3.00	46	23	340	104	99.99	0.03	236.67	49.74
Stat Based	80/20	10	193.63	3.45	45	13	303	54	99.99	0.04	288.89	35.72
Stat Based	90/10	10	248.44	3.94	46	24	325	132	100.00	0.00	363.09	77.28