Extending ASOCS to Training-Set-Style Data

A Thesis
Presented to the
Department of Computer Science
Brigham Young University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Edward F. Hart
August 1992

This thesis, by Edward F. Hart is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the thesis requirement for the degree of Master of Science.

Tony Martinez, Committee Chairman

________________________ Douglas M. Campbell, Committee Member

____

Date Larry C. Christensen, Graduate Coordinator

Acknowledgments

Many thanks to Dr. Tony Martinez for the unceasing efforts to help me think through the problems encountered in understanding the ASOCS neural network.

Dedicated to my father William F. Hart 1931 - 1991, who taught me how to work.

Also many thanks to my wife Darlene and to my children for the countless hours taken away from the family in order to finish this thesis.

Table of Contents

1. Introduction 1

2. Definitions and ASOCS Overview 3

2.1 Incremental Learning vs. Training Examples 3

2.2 Critical vs. Don't Care Variables 4

2.3 Erroneous - Don't Know Data 4

2.4 Architecture 5

2.5 Instances and Instance Sets 7

2.6 Priority ASOCS 13

3. Training Set Applications 22

3.1 Exclusive Or 22

3.2 4 bit Encoder 23

3.3 House Vote Statistics 24

3.4 Zener Diode 26

3.5 Hepatitis Survival 29

3.6 Monk Data Sets 31

4. Problems and Extensions to the basic ASOCS model 34

4.1 Table Terminology 34

4.2 Mutual Exclusion 35

4.3 Arbitrary Discretization 38

4.4 One Difference Minimization with Mutual Exclusion 43

4.5 Missing Variables 47

4.6 Generalization 54

5. Conclusions and Future Research 68

5.1 Theorems and Proofs 68

5.2 Mutual Exclusion 68

5.3 Arbitrary Discretization 69

5.4 One Difference Minimization 69

5.5 Priority ASOCS 69

5.6 Incremental Learning 70

5.7 Missing Data 71

5.8 Generalization 72

5.9 Large Test Sets 72

6. Appendix: Simulator 74

7. Bibliography 80

1. Introduction1. Introduction

Adaptive Self Organizing Concurrent Systems (ASOCS [Mart86]) has been shown to learn rule based applications more efficiently than other neural networks. ASOCS is designed to incrementally learn 'if-then' rules. ASOCS is not efficient for analog and discrete multi-state training-set-style applications for which other neural nets have shown potential. The purpose of this thesis is to show that the basic ASOCS model can be modified to handle training set data that is processed by other neural networks. Data from other neural network style applications is mapped into appropriate data sets and run through an ASOCS model simulator. Extensions to the basic ASOCS model needed to implement efficiently neural net style applications are discovered. These extensions are discussed and potential solutions presented. These extensions are suggestions for improvements to the basic ASOCS model. This thesis is one of analysis and discovery about the ASOCS model, not implementing extensions to the ASOCS model, or providing theorems about the ASOCS model.

In order to understand the extensions discovered in chapter four, chapter two explains the terminology and basics of ASOCS instance set mathematics. This chapter may be omitted if the reader wishes to proceed directly to the training set data. The eight training sets presented in chapter three are samples from a broad set of applications. The goal is not to solve these eight data sets, but to adapt the learning speed and self-adaptability of the ASOCS neural net to handle training-set-style data. In chapter four, problems with the basic ASOCS model are discovered. Extensions to the basic ASOCS neural net are added as chapter four progresses. Various combinations of the extensions are tested with a simulator. By the end of chapter four the extended ASOCS model demonstrates potential in handling training-set-style data. Chapter five is a summary of the extensions to the model. Chapter five also discusses future research. The appendix presents the simulator used to test the data sets with the extensions.

Neural net technology is studied in this thesis because it offers several advantages to von Neumann technology. Parallelism with simple computation nodes and distributed processing appear to permit the processing of large amounts of data [Stev79]. Fault tolerance causes graceful degradation of performance as components fail. The self adaptability of neural nets is an alternative to programming a von Neumann central processor. Neural nets are able to generalize, or learn examples not explicitly taught to the network.

2. Definitions and ASOCS Overview

2. Definitions and ASOCS Overview

In this chapter definitions and the fundamentals of ASOCS instance set dynamics are explained. This is critical to understand the shortcomings of the ASOCS model for handling other neural network training-set-style data. This chapter should be read before chapter four, but may be omitted for chapter three.

2.1 Incremental Learning vs. Training Examples

2.1 Incremental Learning vs. Training Examples

Training-set-style data can be defined as samples or examples of discrete multi-state or analog data which describe the relationship between input data and output data.

There are several differences between if- then rules usually presented to ASOCS models, and the training-set-style data studied in this thesis. The ASOCS neural net model learns rules in an incremental fashion. Older rules are learned as general cases, and then later rules are provided as exceptions to the general cases. When inconsistencies occur, the most recent rule has priority [Mart86]. As an example of incremental style learning, assume a robot is being taught when to open an umbrella. A general rule of *Raining implies Open Umbrella* is presented. Let **R** represent raining and **U** represent an open umbrella. The rule **R** **U** describes the implication. However, when the robot looks outside a window and sees it raining, it opens the umbrella. To correct this, an exception to the general rule is presented: *Not Outside implies Don't Open Umbrella.* Let **O** represent outside. This yields the rule of (**O** **U**) .

However, training-set-style data has no implied precedence for the examples (rather than rules) presented to the model. All examples are of equal value, with no example being of greater importance or priority. For example, let us assume loan approval data is being presented to a neural network to learn good versus bad loans. Presentation order of 100 different loan applications to the neural net is irrelevant.

2.2 Critical vs. Don't Care Variables

2.2 Critical vs. Don't Care Variables

Typically the rules which are presented to the ASOCS model contain only those critical or salient variables necessary to make the rule complete. Variables which do not contribute information to the rule are considered *Don't Care* and are not included. The exception rule for the Robot and Umbrella model presented in section 2.1 is a good example.

Note that the state of raining is not present in the rule of *Not Outside implies Don't Open Umbrella (O U).* If the robot is inside then raining outside does not matter.

Typically training-set-style data presents every variable for each example. Loan application data for all 100 applications would include age, marital status, monthly income, bankruptcy status, and debt ratio, regardless of their relevance in each application.

One of the goals of neural network learning is to discern which features are critical. A feature can be made up of several variables.

2.3 Erroneous - Don't Know Data

2.3 Erroneous - Don't Know Data

Incremental learning deals with missing or erroneous information in older rules by superseding them with newer more accurate rules. This inconsistency resolution depends upon the order of presentation. If the Robot and Umbrella model develops a leaky roof, then the rule of Not Outside would be inconsistent. A new rule is presented: *Raining and Leaky Roof and Not Outside implies Open Umbrella*. Let **L** represent Leaky Roof. Then (**R L O U**) describes the rule. This new exception to the previous rules eliminates the inconsistency.

Training-set-style data may include missing or erroneous data for some of the variables in some of the examples. Variables which are included, but which have no data are called **Don't Know** variables. In contrast, the variables in ASOCS rules which are not included because they are not critical, are called *don't care* variables (section 2.2). In the loan application example, some of the applications may have an incorrect age, or a missing bankruptcy status. The neural net is expected to **generalize** sufficiently to ignore the errant data and still correctly categorize the good versus bad loans.

2.4 Architecture2.4 Architecture

The ASOCS network is made up of connected nodes.

- **Black Box Diagram**

The system accepts discrete variables x_i, and maps them in a parallel fashion to a set of discrete outputs z_i. Feedback from the output goes back to the input for further processing, along with any new inputs (See Figure 1). There are two modes of operation: Learning and Execution.

2.4.1 Learning ModeLearning Mode

In the learning mode, new 'if-then' rules in the form of discrete variable conjunctions, implications and discrete variable outputs (called instances) are presented

- **Single Node**

to the network in an incremental fashion. The network is composed of *nodes* which are interconnected in some fashion. Each node is a storage point for instances, with sufficient computational capability to configure itself with respect to all the other nodes. The internal nodes reconfigure themselves in a parallel and self organizing manner. Learning of an instance has time complexity **log *n*** time, where *n* is the number of internal nodes. The network of nodes maintains a consistent and somewhat minimal configuration. In some models each node has two inputs, a control unit and one output. The output can go to many other nodes (see Figure 2).

2.4.2 Execution ModeExecution Mode

Execution causes no changes to the internal architecture and connections of the network. The network acts as a hardware circuit. Inputs are presented to the network and mapped to discrete variable outputs in log *n* time, in accordance with the incrementally entered instances.

2.5 <u>Instances and Instance Sets</u>2.5 Instances and Instance Sets

2.5.1 Basic DefinitionsBasic Definitions

2.5.1.1 <u>Instance.</u> An **instance** is an implication whose left hand side is a conjunction of discrete input variables and whose right hand side is a discrete output variable. Thus the instance R O L U reads as Rain and not Outside and Leaky Roof imply Umbrella open.

2.5.1.2 <u>Instance Set.</u> An **instance set** is a group of instances.

2.5.1.3 <u>Fulfill.</u> An instance set (network) **fulfills** a set of input variables when it outputs a consistent output variable. This means that the input variables match the input variables of one of the network instances.

2.5.1.4 <u>Polarity.</u> The output variable determines the **polarity** of the instance, **positive** or **negative**.

2.5.1.5 <u>Concordancy.</u> Two instances are **concordant** if they have the same polarity (e.g. R U and R O L U). Opposite polarity makes them **discordant** (e.g. R O U , R O U).

2.5.1.6 <u>Discriminant Variable.</u> If two instances have the same input variable, but of opposite sign in the respective instances, then it is a **discriminant variable** (e.g. R O U , R O U O is the discriminant variable). This discriminant variable provides a means of differentiating the two instances when a set of input variables are presented to the model. For example, if the variables R O are presented to the model, then the first instance would be matched and the output would be U. All discordant instances must have a discriminant variable.

2.5.1.7 <u>New and Old Instances.</u> Order of presentation is fundamental to the ASOCS learning algorithm. The first instance presented to the model is the oldest. When another instance is presented, it becomes the *new instance* (**NI**) and the previous instance is the *old instance* (**OI**). As newer instances are presented, all older instances assume a subordinate priority. The next five definitions are from the OI's perspective.

2.5.1.8 <u>Subset.</u> Let A B   Z be an OI and A B C   Z be the NI. Then the OI is a **subset** to the NI because it maps a smaller number of variables.

2.5.1.9 <u>Superset.</u> Let A B C   Z be the OI and A B   Z the NI. Then the OI is a **superset** to the NI because it maps a larger number of variables.

2.5.1.10 <u>Equal.</u> A B C   Z and A B C   Z are **equal** instances because they have the same input variables.

2.5.1.11 <u>Overlap.</u> A B C   Z and C D E   Z are **overlap** instances because they each have different input variables which are not an improper subset of one another.

2.5.1.12 <u>Discriminated.</u> The NI contains a discriminant variable with respect to the OI.

2.5.2 LearningLearning

There are two aspects to learning: **consistency** and **minimization**. As each instance is incrementally learned, it is presented to the instance set. As each new instance is learned, the network fulfills that instance. New instances (**NI**) always take precedence over old instances (**OI**). The instance set is always made consistent as each NI is learned. The minimization uses only those algorithms which are quick. Consistency is necessary to maintain the integrity of the network, while minimization is desirable for generalization (learning) and parsimony.

2.5.2.1 **<u>Consistency.</u>** Two instances are inconsistent if their respective sets of input variables can be simultaneously matched and they yield contradictory outputs. The input variables need not be the same for both instances. For example, the old instance (**OI**)

A B Z and the NI C D Z are inconsistent because there is no discriminating variable. If the input state for the network were set to A B C D, both instances would match. Consistency is derived in various manners, depending upon the relationship between the two discordant instances.

2.5.2.1.1 Superset. The OI is deleted from the learned instance set when the OI is a superset to the NI. Note that the reference is **always** made from the OI's perspective. For example, assume that these two instances exist:

A B C Z OI deleted from instance set

A B Z NI added as is to instance set

2.5.2.1.2 Equal. The OI is deleted from the learned instance set.

A B C D Z OI

A B C D Z NI

In this example, the OI is deleted from the instance set, and the NI is added to the instance set.

2.5.2.1.3 Subset and Overlap. Discriminant Variable Addition (DVA) is performed on the OI to make the instance set consistent. The OI is removed and replaced with one or more new instances, each containing a discriminant variable from the new instance. This ensures that the instance set is consistent, and the OI still maintains its value.
For example, assume that these two instances exist:

A B C Z OI
C D E Z NI

The NI is overlap. Variables D and E are not in the OI, therefore they can be used for discrimination. After modification the instance set becomes:

A B C D Z OI with variable D added
A B C E Z OI with variable E added

C D E Z NI

In this example, two instances are created from the OI. The first instance adds the negated variable D from the NI. The second instance adds the negated variable E from the NI.

For another example, assume that these two instances exist:

A B Z OI
A B C Z NI

The NI is subset. Variable C is not in the OI; thus it can be used for discrimination. This yields the following modified instance set:

A B C Z new instance from OI

A B C Z NI

Once a NI is consistent with all discordant OI's, it may be minimized against all other concordant instances.

2.5.2.2 <u>Minimization.</u> Minimization serves two purposes. First, it provides *parsimony* to the instance set, reducing the number of instances. Minimization makes a physical implementation of the instance set smaller and process faster. Second, it provides a means for ASOCS to *generalize* or *learn*. This is one of the main goals of neural networks. Minimization retains all pertinent information. Only those kinds of minimization which compute quickly and have simple algorithms are implemented. Minimization occurs only between concordant instances (i.e. instances with the same polarity). There are three basic boolean identities which are used to perform minimization:

1. $x + xy \Rightarrow x$
2. $xy + xy \Rightarrow x$
3. $x + x\,y \Rightarrow x + y$

The resultant minimized instance set has no redundant instances. However, the minimization is not unique. There may exist other minimal instance sets which are different, yet fulfill the same examples during execution.

2.5.2.2.1 Normal Minimization. This type of minimization uses the boolean identity

$x + xy$ x. If any OI is a **subset** or an **equal** instance to the NI, the NI is deleted.

A Z OI 1
B Z OI 2

A B Z NI

In this example the NI could be minimized against either of the two OI's. The old instance set remains unchanged.

If any OI is a **superset** to the NI, the OI is deleted. In this situation, the NI could cause many OI's to be removed.

A D E Z OI 1
A C D Z OI 2
A B Z OI 3

A Z NI

In this example the NI would force all of the OI's to be deleted. The learned instance set would contain only the NI.

2.5.2.2.2 One Difference Minimization. This refers to the case where two concordant instances contain one discriminant variable, and one instance is a subset of the other. **2.5.2.2.2.1 <u>One Difference Equal Minimization</u> uses the boolean identity $xy + xy$ x. When two instances have exactly one discriminant variable, and the same input variables, this is called One Difference Equal Minimization. Both instances are replaced by one instance with the discordant variable removed.**

A B C Z OI 1
A B C Z OI 2

A B C Z NI

In this example the NI can minimize with either OI. The NI and the first OI will minimize to A C Z. Or the second OI and the NI will minimize to A B Z. The NI can not be minimized with both OI's. Hence, as stated above, minimization is not unique.

2.5.2.2.2.2 <u>One Difference Superset Minimization</u> Any OI which is a superset to a NI, and has one discordant variable to the NI, can be minimized. This fact is based upon the boolean identity $x + x y$ $x + y$.

A B C D Z   OI 1
A B E Z   OI 2

A B Z   NI

The first OI and the NI minimize to A C D Z. NI is left intact. NI and the second OI minimize to B E Z. NI remains unchanged and is added to the learned instance set in its original form.

2.5.2.2.2.3 <u>One Difference Subset Minimization</u> An OI which is a subset to a NI, and has one discordant variable to the NI, can be minimized. Only the NI is altered, as in the example below.

A D Z   OI 1
A B D Z OI 2

A B C D Z NI

If the NI and the second OI are minimized, the new NI is A C D Z. Then if it is minimized with the first OI, the final NI is C D Z. Note that if another order of minimization is used the result is different. NI and the first OI yield a new NI of

B C D Z. This modified NI and the second OI would not minimize because they are overlap to one another.

Multiple minimizations are completed through a rebroadcasting process. Rebroadcasting (also called ripple effect) means that a modified NI or OI is presented to the network more than once. Rebroadcasting continues until no instances change. This rebroadcasting process does not yield a unique set of instances, but depends on the order of old instance minimizations against the new instance. However, differing instance sets fulfill the same when used in the execution phase. Rebroadcasting violates log *n* learning. If the rebroadcasting is done during idle time, then the model still maintains fast learning times. Rebroadcasting is used in the simulations developed for this thesis. It is automatically invoked in the simulations whenever One Difference Minimizations occur.

2.6 Priority ASOCS2.6 Priority ASOCS

Another style of the ASOCS model has been developed, called *Priority ASOCS* [Hugh89]. The major difference is that each instance in the learning set has an explicit priority attached to it. The first has a value of 1, the second has 2 and so on. The higher number has greater priority. The explicit priority enforces the rule that each New Instance has higher priority than all Old Instances. The advantage this provides to the learning algorithm is in the elimination of DVA (section 2.5.2.1.3). This will become evident as we go through the various consistency and minimization rules. The same conventions for defining superset, subset, equal, and overlap apply in this model as previously described in the traditional ASOCS model.

2.6.1 LearningLearning

Learning has the same requirements of consistency and minimization.

2.6.1.1 Consistency. This applies to discordant instances. The number after each instance is its priority.

2.6.1.1.1 Superset. The OI is deleted.

NI: A B Z (2)
OI: A B C Z (1)

Delete the OI

2.6.1.1.2 Equal. The OI is deleted

NI: A B Z (2)
OI: A B Z (1)

Delete the OI

2.6.1.1.3 Subset. NI is added, along with its higher priority and the OI remains. This in effect leaves the OI as a general rule (with lower priority) and the NI as an exception. Note that there is no DVA, thus helping to maintain parsimony.

NI: A B C Z (2)
OI: A B Z (1)

Keep both instances

2.6.1.1.4 Overlap. This is identical to subset. The NI is added and its higher priority makes it an exception to the lower priority overlap instances.

NI: A B C Z (3)

OI: B D Z (2)
OI: C D Z (1)

Keep the OI's and NI.

2.6.1.1.5 Discriminated. If there is a discriminant variable between the two instances, there is no removal of the subset, superset or equal OI. A discriminated NI is treated the same as Subset and Overlap. NI is added. Conceptually, discriminated follows the idea of an older general rule and a newer exception. The priority does not have to be incremented because of the discriminant variable.

NI: A B Z (2) priority could be (1)

OI: A B C   Z (1)

Keep the OI and NI

2.6.1.2 <u>Minimization.</u> Once the instance set is consistent, the same types of minimization are performed as in the basic ASOCS model. Minimization is used for parsimony and generalization. Concordant instances (instances having the same output polarity) are compared for potential minimization. There are two types of minimization, Normal and One Difference.

Another concept for minimization must be clarified at this point. Cont_Greater(OI) is a function which means contradictory instance greater than. It is true if there exists an instance which contradicts the New Instance and whose priority is greater than the Old Instance. Formally, Cont_Greater(OI)= true if

{ I   (I contradicts NI) & (PR(I) > PR(OI))} ,

otherwise false.

Informally, it means that the new instance is a subset to the old instance. It may be removed if there is no intervening contradictory instance. If there is a contradictory instance with a priority greater than OI's, and less than NI's, then the NI must be kept as an exception to the contradictory instance. Cont_Greater will become more evident in section 3.3.1.2.3. For example, assume that the learned instance set contains:

OI: A B   Z (1)
OI: A C   Z (3)

If NI A B D   Z (5) is presented (which is a subset to the first OI), then because the second OI contradicts the NI, and has a higher priority than the first OI, the NI is added. If the second OI did not

exist, then the NI would not be added.

2.6.1.2.1 Normal Superset. OI is deleted. For example:

NI: A B Z (2)

OI: A B C Z (1) delete OI

2.6.1.2.2 Normal Equal. Delete OI. For example:

NI: A B Z (2)

OI: A B Z (1) delete OI

2.6.1.2.3 Normal Subset. Add the NI if Cont_Greater(OI) else delete NI. For example:

NI: A B C Z (6)
OI: A B Z (3)

OI: A C Z (5)

with the second OI in the set, include NI, else delete NI. Note that if the contradictory instance had a priority of 2 or 1 (rather than 5), then there would be no need for NI, as it is subsumed by the OI.

2.6.1.2.4 One Difference Superset. Delete the one difference variable from the OI. For example:

NI: A B Z (2)
OI: A B C Z (1)

become

NI: A B Z (2)

OI: A C Z (1) one difference variable B is removed

2.6.1.2.5 One Difference Equal. If Cont_Greater(OI) then Add NI & Modify OI

else Delete OI & Delete One Difference variable from NI & add modified NI.

Case 1 NI: A B C Z (5)
OI: A B C Z (1)

becomes

NI: A C Z (5) and OI deleted.

Case 2 NI: A B C Z (5)
OI: A B C Z (1)
OI: A B Z (3)

The first OI is minimized with NI because the second OI is Cont_Greater. This yields:

NI: A B C Z (5)
OI: A C Z (1) variable B removed

OI: A B Z (3)

2.6.1.2.6 One Difference Subset.

If Cont_Greater(OI) then

Add NI

else

remove One Diff variable from NI *.

if Cont_Greater(OI) then

add NI without modification

else

*.

Case 1 NI: A B C Z (3)
OI: A B Z (1)
OI: A C Z (2)

The NI is added unchanged because the second OI is Cont_Greater().

Case 2 NI: A B C Z (3)
OI: A B Z (1)

The NI is minimized to yield:

NI: A C Z (3) variable B removed
OI: A B Z (1)

Note that in this model, as in the basic ASOCS (discussed in 2.5.2.2.2), minimizing the one difference variable provides the opportunity (but not the requirement) to rebroadcast the minimized instance to the instance set for further minimization. Background processing could perform this function when the network is idle.

2.6.1.3 Equal Priority. In the discussion about the Priority ASOCS the increment of the New Instance priority has been assumed to be 1. This is called normal priority. In fact it is necessary to increment the priority only when needed to avoid an inconsistency. In other words: Leave the priority at the last one used unless the addition of this instance makes the learned instance set inconsistent. For example, OI A B   Z (1) encounters NI of A B C   Z . If the priority of NI is not at least 2, the instance set will be inconsistent. However, if NI is A B   Z , then the discriminant variable B allows the priority of the NI to stay at 1.

The effect of this algorithm change to Priority ASOCS is to reduce the value of the highest priority. However, this does not alter the consistency or minimization algorithms. The learned instance set still fulfills the same during execution phase. For this thesis, incrementing the priority only for inconsistent instances is called Equal priority. It will be shown later in the thesis in sections 4.6.3.3 and 4.6.3.4 that when Mutual Exclusion (defined in 2.6.1.4) is used with training-set-style data, there is always a discriminant variable and thus the priority stays at 1 (hence the notion of equal). The only time the priority increases is when there is an inconsistent instance presented for learning.

2.6.1.4 Mutual Exclusion. All variables used in this discussion have been boolean. They either are positive or negative (A or A). The negative is mutually exclusive of the positive polarity and the combination of positive and negative completely define the set space of the variable. In the next two chapters, training set data is discussed which contains analog data. Some of this data needs to be discretized (subclassified) into more than two classes. There may be any number of subclasses needed to properly represent a variable. There are two approaches to this problem:

Figure 3 – Mutual Exclusion

2.6.1.4.1 Multiple separate variables. If an analog variable needs to be grouped into four subclasses, each subclass can be represented by a different symbol. In this case letters of the alphabet are used. Thus a variable representing temperature which ranges from 0¯ C to 45¯ C might be broken into these subclasses: 0 - 10 , 11 - 20, 21- 39, 40 - 45. The first could be represented by A, the second by B, and so on (see Figure 3). Note that the symbols have no implied relationship to one another, nor are they mutually exclusive, and polarity can still be used. The problem with this approach is that Discriminant Variable Addition occurs a great deal because of all the overlap variables. This leads to very large instance set sizes.

For example, let OI A G K Z and NI B C F Z be two instances representing some analog examples. In order to make the instance set consistent, the OI must be modified to become 3 instances, each one containing one of the overlap variables from the NI:

A G K B Z
A G K C Z
A G K F Z

These three instances allow the OI to be discriminant with all three overlap variables in the NI. This ensures consistency.

Thus, the DVA growth will increase according to the number of overlap variables.

2.6.1.4.2 Mutually Exclusive Subclasses. The same analog variable can be classified using the same subranges in such a way that the subranges are mutually exclusive. The variable is described by a Major Class represented by a symbol (e.g. T for temperature), and a minor class which is represented by multiple symbols, one per subrange (e.g. a b c d). Thus the temperature variable is described as T:a for the first subrange, T:b for the second and so on. Note that the subranges are mutually exclusive because only one minor class is allowed with the major class. This technique of representing more than two subclasses is called Mutual Exclusion. The variables are sometimes called multi-state inputs.

2.6.1.5 <u>Polarity versus Mutual Exclusion.</u> Both solutions to the subclassification (discretization) problem permit polarity. This creates confusion when more than two subclasses are used. For example, T:a means temperature not between 0¯ and 10¯. It could be in subclass b, c, or d. In order to avoid this confusion, the policy of no polarity for mutually exclusive variables (more than 2 discretizations) is followed. Also, for the sake of uniformity, if at least one input variable to an instance needs to use Mutual Exclusion, then all input variables must use Mutual Exclusion. Thus A and A are represented as A:a and A:b.

3. Training Set Applications3. Training Set Applications

This chapter presents the eight data sets used to test the

ASOCS instance set model. The first two are included to show that ASOCS can easily handle the classical test cases. The last six sets demonstrate weaknesses (discussed in chapter 4) in the ASOCS model for handling training set data. Raw data is gathered for each data set. Then the data is mapped into ASOCS instances.

In order to run the data through the ASOCS simulator, several different mappings are done. The mappings yielding the best recognition for each data set are shown in chapter 4.

3.1 Exclusive Or3.1 Exclusive Or

The first set of 2 input Exclusive Or data is very small, but difficult for many classical neural nets to learn. For example, the back propagation model takes several thousand repetitions of the training set to generalize. The back propagation model can get stuck in local minima [McCl86] pp 331 - 334. ASOCS handles it easily. The two input exclusive or data set is:

0	0	0
0	1	1
1	0	1
1	1	0

Figure 3.1 Exclusive Or Training Set

Figure 3.2 is the complete training set mapped for ASOCS. A stands for the first variable, B stands for the second variable. The apostrophe marks a negated versus a non-negated variable. This training set has 100% recognition. This example is included to show that ASOCS handles the classical neural net test case easily.

A	B	Z
A	B	Z
A	B	Z
A	B	Z

Figure 3.2 2 Input Exclusive Or mapped for ASOCS

3.2 4 bit Encoder 3.2 4 bit Encoder

The second set of Encoder data is also small, and difficult for other neural nets to learn [McCl86] pp 335 - 339. The Boltzmann machine takes many thousands of iterations to solve this problem. The chosen sample training set shown below comes from McClelland:

0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
1	1	1	1	0
1	1	1	0	1
1	1	0	1	1
1	0	1	1	1
0	1	1	1	1
0	0	1	1	0
0	1	1	0	0
1	1	0	0	0
1	0	0	1	0
1	0	1	0	0
0	1	0	1	0

Figure 3.3 4 bit Encoder Training Set

Figure 3.4 is the complete training set mapped for ASOCS. This training set maps like the previous data set; each letter is the respective variable. This training set has 100% recognition. Once again, it is included only to show that ASOCS learns one of the classical test cases easily.

A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z
A	B	C	D	Z

A B C D Z
A B C D Z
A B C D Z

Figure 3.4 Encoder Data mapped for ASOCS

3.3 <u>House Vote Statistics</u>3.3 House Vote Statistics

The third data set is House Vote Statistics from 1984 Congressional voting on 16 issues [Schl87]. Jeff Schlimmer tested the data using an incremental conceptual clustering network model to learn political party voting bias. He got 90 - 95% correct classification. This data set uses 16 voted issues to predict by the voting pattern whether the politician is a Republican or a Democrat. It is obtained from the Irvine Computer Learning Database. There are many missing votes. They may mean several things, but are simplified to one type in the raw data set. Of the 435 instances, 267 are democrats and 168 are republicans. All 16 input variables take on one of 3 values: yes, no, unknown. The single output variable is Republican or Democrat. The attributes of this voting data set are:

Output. Class Name: (democrat, republican)
1. **handicapped-infants: (y,n)**
2. **water-project-cost-sharing: (y,n)**
3. **adoption-of-the-budget-resolution: (y,n)**
4. **physician-fee-freeze: (y,n)**
5. **el-salvador-aid: (y,n)**
6. **religious-groups-in-schools: (y,n)**
7. **anti-satellite-test-ban: (y,n)**
8. **aid-to-nicaraguan-contras: (y,n)**
9. **mx-missile: (y,n)**
10. **immigration: (y,n)**
11. **synfuels-corporation-cutback: (y,n)**
12. **education-spending: (y,n)**
13. **superfund-right-to-sue: (y,n)**
14. **crime: (y,n)**
15. **duty-free-exports: (y,n)**
16. **export-administration-act-south-africa: (y,n)**

Figure 3.5 Voting Data Variable Description

Below is a sample of the raw data.

republican,n,y,n,y,y,y,n,n,n,y,?,y,y,y,n,y
republican,n,y,n,y,y,y,n,n,n,n,n,y,y,y,n,?
democrat,?,y,y,?,y,y,n,n,n,n,y,n,y,y,n,n
democrat,n,y,y,n,?,y,n,n,n,n,y,n,y,n,n,y
democrat,y,y,y,n,y,y,n,n,n,n,y,?,y,y,y,y
democrat,n,y,y,n,y,y,n,n,n,n,n,n,y,y,y,y
democrat,n,y,n,y,y,y,n,n,n,n,n,n,?,y,y,y
republican,n,y,n,y,y,y,n,n,n,n,n,n,y,y,?,y
republican,n,y,n,y,y,y,n,n,n,n,n,y,y,y,n,y

Figure 3.6 Sample Votes in Original Data

All variables are discretized using Mutual Exclusion. All 16 input variables are discretized three ways, y n and ?. Republican is positive output, Democrat is negative (no bias is intended or implied).

```
A:n  B:y  C:n  D:y  E:y  F:y  G:n  H:n  I:n  J:y  K:?  L:y  M:y  N:y  O:n
                                 P:y      V
A:?  B:y  C:y  D:?  E:y  F:y  G:n  H:n  I:n  J:n  K:y  L:n  M:y  N:y  O:n
                                 P:n      V
A:y  B:y  C:y  D:n  E:y  F:y  G:n  H:n  I:n  J:n  K:y  L:?  M:y  N:y  O:y
                                 P:y      V
A:n  B:y  C:n  D:y  E:y  F:y  G:n  H:n  I:n  J:n  K:n  L:n  M:?  N:y  O:y
                                 P:y      V
A:n  B:y  C:n  D:y  E:y  F:y  G:n  H:n  I:n  J:n  K:n  L:y  M:y  N:y  O:n
                                 P:y      V
A:n  B:y  C:n  D:y  E:y  F:n  G:n  H:n  I:n  J:n  K:?  L:?  M:y  N:y  O:n
                                 P:n      V
```

Figure 3.7 Voting Data for ASOCS

The first line of figure 3.7 is a republican who voted No on handicapped infants (A), Yes to water project cost sharing (B), No to adoption of the budget resolution (C), etc..

3.4 <u>Zener Diode</u>3.4 Zener Diode

The fourth data set is the Zener Diode data obtained from Hill Air Force Base at Ogden Utah from Dr. Lloyd Allred [Allr88]. Zener diodes are used to protect the integrated circuitry on computer boards which control guided missiles. If the diodes take a large voltage surge, then they may be damaged. Current detection of faulty diodes is done manually with an oscilloscope. Dr. Allred used a Bayesian Classifier network to classify the diodes into 6 classes: one good and five bad. He obtained 90% recognition. Each instance consists of twelve input variables, all of which are analog values, as follows:

1. minimum voltage,
2. maximum voltage,
3. through 12 are various slope calculations of a waveform.

The output variable is a digit from 0 to 5, indicating the classification of a diode. The classifications are:

0: Good,
1: Bad Amp,
2: Open Bottom,
3: Open Top,
4: Short Bottom,
5: Short Top.

Only the first class is acceptable, the others are bad. There are 68 samples in the training set used for learning and 51 samples in the training set used for execution. Below is a sample of the data.

-.860	.88	.191	.315	.364	.679	.802	.187	.331	.394	.662	.812	BadAmp	1
-.925	.92	.175	.300	.344	.694	.831	.153	.259	.282	.747	.841	Good	0
-1.120	1.04	.187	.325	.381	.669	.806	.154	.243	.278	.751	.846	OpenBottom	2
-1.165	.98	.182	.312	.359	.682	.812	.175	.299	.351	.695	.825	OpenBottom	2
-1.115	.97	.188	.345	.418	.667	.812	.182	.296	.346	.704	.818	OpenBottom	2

```
 -.020 1.06  .273  .364  .636  .818  .909  .168  .289  .335  .711  .838    ShortBottom
4
-1.020 1.02  .184  .323  .361  .696  .823  .156  .263  .305  .736  .844     Good
0
-1.080 1.20  .161  .286  .323  .708  .826  .183  .311  .384  .683  .817    OpenTop
3
 -.900 1.00  .160  .265  .296  .735  .846  .176  .297  .333  .703  .824 13 Good
0
```

Figure 3.8 Sample Zener Diode Data

For example the first line means that the minimum voltage is -.860, the maximum voltage is .88, the first slope measurement has a voltage of .191 etc., and the resultant classification is a Bad Amp.

This training set is mapped using both Mutual Exclusion and Non-Mutual Exclusion.

3.4.1 Without Mutual ExclusionWithout Mutual Exclusion

In this set each of the variables is discretized into four subclasses. Four unique characters represent each variable. The first four letters of the alphabet in lower case are for the first variable (a, b, c, d). The next four letters are for the second variable (e, f, g, h). This pattern continues for each of the other 10 input variables. The output variable uses polarity to distinguish good diodes from bad diodes. Negative polarity marks all five classes of bad diodes, positive polarity marks the class of good diodes.

```
b g j n r w B C H L Q V     a
a g j n r w B C H L Q V     a
d g k o t x B C H L Q V     a
```

Figure 3.9 Zener Diode Data Mapped to ASOCS

The first entry in figure 3.9 has the minimum voltage in the second subrange b (a, b, c, d are used for this variable), maximum voltage in the third subrange g (e, f, g, h), the first slope in the second subrange j (i, j, k, l) and so on. The output value of a means that the sample is for a bad diode.

3.4.2 With Mutual ExclusionWith Mutual Exclusion

In this set, each variable is assigned a major class and then broken into mutually exclusive subclasses. In figure 3.10 each variable is discretized into four. The six types of diodes are represented by subclasses a - f. a is good, with the other letters representing bad amp etc. in the same order as the raw data listing of bad types.

```
A:b  B:c  C:b  D:b  E:b  F:c  G:d  H:a  I:b  J:b  K:c  L:d     a:b
A:b  B:c  C:b  D:b  E:b  F:c  G:d  H:a  I:b  J:b  K:d  L:d     a:a
A:b  B:c  C:b  D:b  E:c  F:c  G:d  H:a  I:b  J:b  K:d  L:d     a:c
```

Figure 3.10 Zener Diode Mutual Exclusion Mapping

The first example in figure 3.10 has minimum voltage denoted by the symbol A with the subclass of b (a, b, c, d are the four possible). Maximum voltage is B in the third subrange c (the same characters can be reused because they are mutually exclusive within the range of the respective variables). The first voltage slope is C in the second subrange b, and so on.

3.5 Hepatitis Survival3.5 Hepatitis Survival

The fifth data set is Hepatitis Survival data [Gong88]. It is provided by Gail Gong from research data obtained in Yugoslavia. The data set uses 19 variables which describe the medical condition of a patient to see if a correct classification can be made as to whether the patient lives or dies. Between 80 - 83% recognition rates are achieved by Gail Gong. This data set is also obtained from the Irvine Computer Learning Database. There is 1 output variable and 19 input attributes:

Class: DIE, LIVE. This is the only output variable.
1. AGE: 10, 20, 30, 40, 50, 60, 70, 80;
2. SEX: male, female;
3. STEROID: no, yes;
4. ANTIVIRALS: no, yes;
5. FATIGUE: no, yes;
6. MALAISE: no, yes;
7. ANOREXIA: no, yes;
8. LIVER BIG: no, yes;
9. LIVER FIRM: no, yes;
10. SPLEEN PALPABLE: no, yes;
11. SPIDERS: no, yes;

12. ASCITES: no, yes;
13. VARICES: no, yes
14. BILIRUBIN: 0.39, 0.80, 1.20, 2.00, 3.00, 4.00
15. ALK PHOSPHATE: 33, 80, 120, 160, 200, 250
16. SGOT: 13, 100, 200, 300, 400, 500,
17. ALBUMIN: 2.1, 3.0, 3.8, 4.5, 5.0, 6.0
18. PROTIME: 10, 20, 30, 40, 50, 60, 70, 80, 90
19. HISTOLOGY: no, yes

Figure 3.11 Hepatitis Variable Description

There are many missing values in the data. They are denoted by '?'. Of the total 155 samples, 32 died and 123 lived. Here is a sample of the raw data. The first line is the first letter of each attribute, which is useful for correlating the data.

```
C a   s s a f m a l l s s a v  bil al   s  alb p
his
2,30,2,1,2,2,2,2,1,2,2,2,2,2,1.00,85,  18,4.0,  ?,1
2,50,1,1,2,1,2,2,1,2,2,2,2,2,0.90,135,42,3.5,  ?,1
2,78,1,2,2,1,2,2,2,2,2,2,2,2,0.70,96,  32,4.0,  ?,1
2,31,1,?,1,2,2,2,2,2,2,2,2,2,0.70,46,  52,4.0,80,1
2,34,1,2,2,2,2,2,2,2,2,2,2,2,1.00,?  ,200,4.0,  ?,1
2,34,1,2,2,2,2,2,2,2,2,2,2,2,0.90,95,  28,4.0,75,1
1,51,1,1,2,1,2,1,2,2,1,1,2,2,?    , ?,   ?,   ?, ?,1
2,23,1,2,2,2,2,2,2,2,2,2,2,2,1.00, ?,   ?,   ?, ?,1
2,39,1,2,2,1,2,2,2,1,2,2,2,2,0.70,  ?, 48,4.4,  ?,1
2,30,1,2,2,2,2,2,2,2,2,2,2,2,1.00,  ?,120,3.9,  ?,1
2,39,1,1,1,2,2,2,1,1,2,2,2,2,1.30,78,  30,4.4,85,1
```

Figure 3.12 Sample Hepatitis Data

The first line of figure 3.12 means that the patient lives (2), is thirty years old (30), is female (2), does not take steroids (1), has antivirals (2), fatigue (2), malaise (2), anorexia (2), a small, firm liver (1,2), palpable spleen (2), spiders (2), ascites (2), varices (2), a bilirubin count of 1.00, alkaline phosphate of 85, sgot of 18, albumin of 4.0, unknown protime (?), and no histology (1).

3.5.1 With Missing Values using Mutual ExclusionWith Missing Values using Mutual Exclusion

The nineteen input variables are all mapped using Mutual Exclusion, even though 13 of them are boolean. The first, and the 14th through 18th input variables have higher numbers of discretization (see the variable list figure 3.11 for the break points used). Because the output is boolean, polarity is adequate to denote living versus dying. Any missing values are marked by a ?.

```
A:b B:a C:b D:a E:a F:a G:a H:b I:a J:a K:a L:a M:a N:b O:b P:a Q:b R:? S:b    L
A:d B:b C:a D:a E:b F:a G:a H:a I:a J:a K:a L:a M:a N:a O:b P:a Q:b R:? S:b    L
A:b B:b C:a D:a E:a F:a G:a H:a I:a J:a K:a L:a M:a N:b O:? P:c Q:a R:? S:b    L
A:c B:b C:b D:a E:b F:a G:b H:a I:a J:b K:b L:a M:a N:? O:? P:? Q:? R:? S:b    L
A:b B:b C:a D:a E:b F:a G:a H:a I:b J:a K:a L:a M:a N:a O:? P:a Q:a R:? S:b    L
A:b B:b C:b D:b E:a F:a G:a H:b I:b J:a K:a L:a M:a N:b O:a P:a Q:b R:e S:b    L
```

Figure 3.13 Hepatitis Data for ASOCS

The first example in figure 3.13 is a patient who lives with his age between 20 and 30 (A:b), sex is male (B:a), yes to steroids (C:b), no to antivirals (D:a) etc..

3.5.2 Without Missing ValuesWithout Missing Values

The only difference with this set from the previous is that any instance with missing values is discarded. This cut the number of instances in half. This grouping is used in chapter 5 for a discussion on missing data.

```
A:c B:b C:a D:a E:a F:a G:a H:a I:a J:a K:a L:a M:a N:b O:a P:a Q:c R:c S:b    L
A:b B:b C:b D:a E:b F:b G:a H:a I:a J:a K:a L:a M:a N:b O:b P:a Q:b R:f S:b    L
A:d B:b C:b D:a E:b F:a G:a H:b I:b J:a K:a L:a M:a N:b O:a P:a Q:b R:f S:b    L
A:b B:b C:b D:b E:b F:b G:a H:a I:b J:a K:a L:a M:a N:c O:d P:a Q:b R:c S:b    L
A:b B:a C:b D:a E:a F:a G:a H:a I:b J:a K:a L:a M:a N:a O:b P:a Q:b R:d S:b    L
A:b B:b C:a D:a E:b F:b G:b H:a I:a J:b K:a L:a M:a N:b O:c P:b Q:b R:f S:b    L
```

Figure 3.14 Hepatitis Data, no Missing Variables

3.6 Monk Data Sets3.6 Monk Data Sets

The last three training sets come from a combined study of learning algorithms, published by Carnegie Mellon University in December 1991 [Carn91]. They are called the MONK's problems. Each example in all three training sets has six discrete-valued input variables. The output variable is either 1 or 0. The attributes have the following subclassifications:

attribute#1 : {1, 2, 3}

attribute#2 : {1, 2, 3}
attribute#3 : {1, 2}
attribute#4 : {1, 2, 3}
attribute#5 : {1, 2, 3, 4}
attribute#6 : {1, 2}

Thus, the six attributes span a space of 432 = 3x3x2x3x4x2 examples.

The "true" concepts underlying each MONK's problem are given by:

MONK-1: (attribute#1 = attribute#2) or (attribute#5 = 1)

MONK-2: (attribute#n = 1) for EXACTLY TWO choices of n (in {1,2,...,6})

MONK-3: (attribute#5 = 3 and attribute#4 = 1) or

(attribute#5 != 4 and attribute#2 != 3)

MONK-3 has 5% additional noise (misclassifications) in the training set.

Here is a sample of the data from the MONK-1 training set:

data_37:	1 1 2 2 3 1	1
data_39:	1 1 2 2 4 1	1
data_42:	1 1 2 3 1 2	1
data_50:	1 2 1 1 1 2	1
data_51:	1 2 1 1 2 1	0
data_53:	1 2 1 1 3 1	0
data_56:	1 2 1 1 4 2	0

Figure 3.15 Sample Monk-1 Data

All three training sets use a complete set of their respective 432 examples for testing the learned instances. MONK-1 has a random selection of 124 instances for the learning phase, MONK-2 has 169 instances for learning, and MONK-3 has 122 instances. In the article in [Carn91], these data sets are tested against 15 learning models, with varying degrees of success.

All three training sets are mapped into both mutually exclusive and non-mutually exclusive training sets. Below are two examples of the same data from MONK-1:

```
6   a d h i m q    V
7   a d h j n p    V
8   a d h j o p    V
9   a d h k l q    V
10  a e g i l q    V
11  a e g i m p    V'
12  a e g i n p    V'
```

Figure 3.16 Monk-1 Data Non-Mutual Exclusion

The first line of figure 3.16 means that the 6th data sample has attribute 1 as 1 (a of a, b, c),attribute 2 as 1 (d of d, e, f), attribute 3 as 2 (h of g, h), attribute 4 as 1 (i of i, j, k), attribute 5 as 2 (m of l, m, n, o), and attribute 6 as 2 (q of p, q) for an output of 0.

```
6   A:a  B:a  C:b  D:a  E:b  F:b     V
7   A:a  B:a  C:b  D:b  E:c  F:a     V
8   A:a  B:a  C:b  D:b  E:d  F:a     V
9   A:a  B:a  C:b  D:c  E:a  F:b     V
10  A:a  B:b  C:a  D:a  E:a  F:b     V
11  A:a  B:b  C:a  D:a  E:b  F:a     V'
12  A:a  B:b  C:a  D:a  E:c  F:a     V'
```

Figure 3.17 Monk -1 Data Mutual Exclusion

Figure 3.17 has the same data mapped as figure 3.16, so the translation of A:a etc. follows the same pattern. This same procedure applies to the Monk-2 and Monk-3 training data sets.

4. Problems and Extensions to the basic ASOCS model4.

Problems and Extensions to the basic ASOCS model

At the start of chapter 2 several differences between training-

set-style data learning and incremental rule learning are discussed. ASOCS is not designed to handle training-set-style data. In this chapter, it is suggested how ASOCS can be extended to deal with training set data. Five problems are discussed in this chapter. Not all of the problems have a solution provided in this thesis. Six extensions are discussed. The extensions to the basic ASOCS model demonstrate the ability to handle training set data.

4.1 Table Terminology4.1 Table Terminology

Throughout this chapter there are many tables used to illustrate and clarify the issues discussed. The data in these tables is helpful in understanding the problems and extensions to the basic ASOCS neural network. These tables contain the results of running the simulator with the various training data sets. The columns are labeled and described as follows:

- Compute Time. The number of seconds and minutes to complete this test run.

- # Learned Instances. How many of the original training set instances remain, along with any DVA instances or modified training instances. This is an indication of the ability of the algorithm to perform generalization or parsimony on the learned instance set.

- # Instances Recognized. This column lists the number of instances (training set examples) in the test set which are correctly recognized (matched) when the model is in execution mode. This recognition comes in two forms:

1. Exact Match. This is the only form used until the generalization section (4.6). This means that the model finds an instance in its learned instance set which exactly matches all input variables with

the test instance. These matchings may be equal or subset.

2. Close Recognition. This is explained in section 4.6.

- Number of Variables. The number of input variables in each instance. Note that most training set data usually use the same variables for all examples. In the case of the six training sets used in this thesis, all of them use all of the variables provided in the raw test data.

Note that in the generalization section 4.6 there are several other column types. These are explained in that section.

4.2 <u>Mutual Exclusion</u>4.2 Mutual Exclusion

4.2.1 Zener DiodeZener Diode

This Mutual Exclusion problem appears when working with the fourth set of data (Zener Diode). There is no way to map a variable into more than two subclasses without Mutual Exclusion[1]. A and A (True and False) are the limit of discretization without going to a subclass. There are no instances matched when the Zener diode data is tested using the basic ASOCS model (Figure 3.9 where different variables are used for discretization). The failure to match is due to the fact that each variable represents a completely separate domain to the ASOCS model. There is no implied mutual exclusivity between variables A B C and D, even though they may be used to represent minimum voltage. Discriminant Variable Addition which makes the instance set consistent, causes the number of instances to grow

[1] a variable with four values could be mapped into A B, A B , A B, A B . However, this introduces the inefficiency of more variables. Also, One Difference Minimization of the instance becomes difficult when each analog variable is represented by two discrete variables.

rapidly. Large instance size is the primary cause for the indeterminate calculation times shown in Figure 4.1.

With the use of a multi-state variable, the Mutual Exclusion problem is solved. Each subclass automatically implies mutual exclusion because if it is in a, it can not be in b or c. In order to handle consistency of discordant instances and minimization of concordant instances, polarity is never combined with Mutual Exclusion (as discussed in 2.6.1.5). For example, a variable with only 2 subclasses (A and A) is represented as A:a and A:b. Another example, for a three state variable Z, Z:c means Z:a and Z:b. This is undesirable for the consistency and minimization algorithms, and is avoided.

Each variable in the Zener Diode set is evenly discretized in all tests. The first set (figure 3.9) has binary outputs, with good versus bad as positive and negative polarity. Using these techniques, the best simulator results on the Zener Diode data are 36 matched correctly out of 51 using 4 subclasses. This concurs exactly with Lloyd Allred's results of 36 out of 51 classified with 100% accuracy. Lloyd Allred's results mean that the Bayesian Classifier matches with no margin of error 36 out of the 51 examples. The other 15 examples have some degree of uncertainty in the determination of an answer. The uncertainty is due to mismatches with some of the slope variables. Of the 68 training instances, 43 remain in the learned instance set.

The results are much poorer when tested with different numbers of subclasses (this is explained in section 4.3). Also, no One Difference minimization is used, as another problem occurs with it.

This is explained in the One Difference section (4.4).

The next test runs use Mutual Exclusion for the output variable as well. The best success is once again with 4 subclasses. Matches are still 36 out of 51 test cases. There is one Bad Amp recognized as an Open Bottom. However, the raw data shows that the differences in the analog values are so small as to make the classification rather arbitrary. Bad Amp and Open Bottom have very close minimum and maximum voltages.

The empirical data tested on the simulator suggests that this extension is useful.

4.2.2 Voting DataVoting Data

The voting data has no analog values in it. Each variable is broken into three

subparts, *yes*, *no* and *don't know*[1]. With no Mutual Exclusion, the problem is not solvable using the simulator. The simulator's results improve when multi-state variables are used.

When Mutual Exclusion is used, the recognition is higher (6 out of 47), number of instances in the learned instance set are smaller (301), and execution time is drastically shorter (50 minutes). The poor recognition rate of 6 out of 47 shows that the basic ASOCS neural network with this one enhancement is still not robust enough to make it useful for this training data set. However, the Mutual Exclusion feature is a step in the right direction.

4.2.3 HepatitisHepatitis

The hepatitis data set is not attempted in a separate variable

---

1 Simply removing the unknown variables and treating them as *don't care* is another alternative. This is explored in the missing data section (4.5)

scenario (Non-Mutual Exclusion). The large number of variables (19) makes it a more difficult problem to solve than the voting data set. For example, if each of the 19 variables are divided into 4 subclasses, the total number of combinations is 4^{19} or 2.74878 $*10^{11}$. 139 instances out of 2.7487 $*10^{11}$ total instances possible, or 5.05694$*10^{-8}$ %, is an extremely small sample size. This points out a difficulty of ASOCS dealing with large numbers of variables. However, this is not the only problem with the Hepatitis data set. Other problems are discussed in other sections of this chapter.

The Mutual Exclusion simulation runs show 0 success. The execution time is reasonable, and the 139 training instances produced 139 instances in the learned instance set. In other words, the model can not perform any minimization or generalization (learning) on the training set.

	Basic ASOCS	Mutual Exclusion				
Data Set	Compute Time	# Learned Instances	Compute Time	# Learned Instances	# Instances Recognized Variables	Number
Zener Diode	Never Finished	> 1,000	1.9 minutes	43/68	36/51 12	
Vote	Never Finished	> 1,000	50.45 min	301/388	6/47 16	
Hepatitis	Never Finished	> 1,000	11.22 min	139/139	0/16 19	

Figure 4.1 Mutual Exclusion Test Runs

These tests suggest that the Mutual Exclusion enhancement helps the basic ASOCS model deal with training-set-style data by elimintating DVA, but is not sufficient to handle the sets well.

4.3 Arbitrary Discretization4.3 Arbitrary Discretization

Figure 4.2 - Zener Diode Min & Max Voltage

Simple discretization routines are used to map the training set data into ASOCS instances. They read in the raw data and evenly divide the range of each variable into the desired number of subclasses. However, the distribution of values over the range of a variable are rarely evenly distributed. Consequently the subclasses do not properly represent good classification of the variable. Figure 4.2 shows the minimum and maximum voltages for Zener Diode. They are not evenly distributed, but tend to cluster in certain portions of the range. The four subclasses marked in the figure are not the best that is being accomplished by a more intelligent clustering mechanism [Bert92]. In neural network research the topic of unsupervised classification, or spontaneous learning addresses this issue [Koho82], [Kosk87], [McCl86], [Sutt81a]. Currently neural network research is being done at Brigham Young University to address this issue [Bert92]. It is beyond the scope of this thesis to solve this problem. It is also well researched in classical statistical analysis [Allr88].

Another problem with classification (in general as well as with ASOCS) is that sometimes the difference between two classes is very small, leaving little room for error. In the Zener Diode case, there are many diodes which are marginal, having very small differences between an acceptable voltage and a bad one. Manual classification of these marginal cases can be arbitrary and debatable. This only adds to the difficulty in performing correct classifications.

In order to verify that a solution for this problem is useful, a manual discretization on the Zener Diode data set is performed. The first two variables of minimum and maximum voltage (see figures 4.4, 4.5) are discretized into 3 subclasses and the rest of the variables are put into one class each (in effect rendering their information value useless, or removing them from the set). The breakpoints for the discretization on the two variables were placed on the limits of the good diode voltages.

This somewhat optimal configuration is achieved only after several iterations and test runs. This points out that training set data has errors and noise which may have to be resolved by an iterative approach. However, non-iterative algorithms could be derived.

The first test uses binary values for the output variable, so that good diodes and bad diodes are in two sets. The run time is 36 seconds, with 9 instances in the learned instance set, and all test sets are recognized (see Figure 4.3). Figure 4.4 is a sample of the training set, and figure 4.5 a complete list of the learned instance set. Note that only one instance defines the good diodes. This is because the discretization grouped them into one set, with both variables being subset b when the diode is good. All other combinations of the three subclasses for both voltage variables ($3^2 - 1 = 8$) are bad diodes. The term *smart* in table 4.3 means that the subclassification of the max and min voltage variables is preprocessed to a more optimal level than merely evenly subdividing the range of the variable.

Training Set (all Zener Diode)	Compute Time	# in Learned Set out of 68	# Matched out of 51	# Matched Incorrectly
Normal Binary	110 secs	43	36	0
Normal Mutual Excl	112 secs	43	36	1
Smart Discrete Binary	36 secs	9	51	0
Smart Discrete Mutual Excl	38 secs	9	51	7

Figure 4.3 Smart Discretization Tests

A:a B:b C:c D:c E:c F:c G:c H:c I:c J:c K:c L:c a'
A:c B:c C:c D:c E:c F:c G:c H:c I:c J:c K:c L:c a'

A:b	B:a	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'
A:b	B:b	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a
A:b	B:a	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'

Figure 4.4 - Sample Training Set

A:a	B:a	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'
A:b	B:c	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'
A:a	B:c	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'
A:c	B:c	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'
A:b	B:a	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'
A:c	B:b	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'
A:a	B:b	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'
A:b	B:b	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a
A:c	B:a	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a'

Figure 4.5 - Complete Learned Instance Set

The next set of data (figures 4.6, 4.7) also uses Mutual Exclusion on the output variable. There are still only 9 instances in the learned instance set, and simulator execution takes 38 seconds. Although all 51 test instances are recognized, there is some confusion on recognizing Short Bottom versus Bad Amp and Open Bottom versus Open Top. The confusion is due to the similarities in the voltage output patterns for these types, and the loss of information by not using the other ten variables. Nevertheless, there is no incorrect recognition of a good versus a bad diode.

A:a	B:b	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:c
A:c	B:c	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:e
A:b	B:a	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:f
A:b	B:b	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:a
A:b	B:a	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:f

Figure 4.6 - Sample Training Set

A:c	B:c	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:e
A:a	B:a	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:f
A:b	B:c	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:d
A:a	B:c	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:c
A:c	B:a	C:c	D:c	E:c	F:c	G:c	H:c	I:c	J:c	K:c	L:c	a:b

A:b B:a C:c D:c E:c F:c G:c H:c I:c J:c K:c L:c   a:f
A:c B:b C:c D:c E:c F:c G:c H:c I:c J:c K:c L:c   a:e
A:a B:b C:c D:c E:c F:c G:c H:c I:c J:c K:c L:c   a:c
A:b B:b C:c D:c E:c F:c G:c H:c I:c J:c K:c L:c   a:a

Figure 4.7 - Complete Learned Instance Set

This manual (smart) classification suggests that there is room for improvement utilizing the ASOCS neural network by providing some sort of intelligent preprocessing and classification.

Note that the two variable test runs on the Zener Diode data are an example of the Critical Variable concept discussed in section 2.2. One of the goals of neural networks is to learn which variables are the most important.

4.4 One Difference Minimization with Mutual Exclusion

4.4 One Difference Minimization with Mutual Exclusion

In chapter 2 we discussed One Difference Minimization. This is an important feature of the ASOCS model to help perform generalization and learn the important features of the instances presented to the network. Without Mutual Exclusion, there is no problem with completely removing the one different variable. With Mutual Exclusion, the situation becomes more complex. Let A:b C:a Z and A:c C:a Z be two instances which attempt One Difference Equal minimization. From section 3.2.2.2.2, the second instance is removed and the A variable is minimized from the first instance, yielding C:a Z. The removal of the A variable is erroneous because it contains more information than just A:b and A:c. The A:a subclass is not available for discrimination, thus information is lost. The One Difference Minimization problem exists in all of the last six data sets.

4.4.1 Modify One Difference AlgorithmModify One Difference Algorithm

In order to solve the One Difference Minimization shortcoming, the one difference minimization routine is modified. Instead of removing the variable, its subclasses are combined, and the modified instance is larger. Using the previous example, the A:b and A:c are not removed, but combined into A:(b or c). The new instance now becomes : A:(b or c) C:a Z.

This extension to the basic ASOCS model increases its time and space complexity. The processing speed is slightly slower and the ASOCS internal architecture expands. The consequence of repeated use of this enhancement is to remove the variable when all of its subclasses minimize into one instance. For example, if A(a or b or c) completely maps the set that A describes, then the original One Difference Minimization technique would cause A to be removed from the instance. It probably requires a large number of training instances to have the minimization process remove a completely mapped variable. None of the eight training sets studied in this thesis has more than 388 training instances, which is far too few to verify that a variable would be removed.

Here is an example of this extension in the Voting Data learned instance set.

```
A:(n y) B:n C:y D:n E:n F:n G:y H:y I:y J:n      K:n L:n M:n N:n O:y P:y
V
A:n     B:n C:n D:y E:y F:y G:n H:n I:n J:(n y) K:n L:y M:y N:y O:n P:n
V
```

Figure 4.8 One Difference Minimization These two instances maintain the correct mapping for the learned instance set, provide some parsimony to the instance set and help with generalization (learning).

There is an important consequence of this minimization technique. It appears that a variable with multiple subclasses is not equal to the individual subclasses from which it is built. For example, A(a or b) is not equal to A:a or A:b, but is a superset to them for the purposes of learning and further minimization. Another way to look at this is to say that A(a or b) is a discriminant variable compared to A:a or A:b. However, once learning stops and execution begins, the multi-state variable will correctly recognize any of its individual subclasses. For example, A(a or b) will recognize A:a or A:b. This subclass inequality results in some Subset or Superset minimization, rather than all Equal minimization. For example, OI A:(a or b) is a superset of NI A:a. Minimization rules from section 2.6.1.2.1 would have the OI removed. In order to prevent the loss of information from this removal, another rule is added: *No minimization of a variable containing more than one subclass can occur until the entire set of subclasses is contained in that variable*. For example, if variable A has 3 subclasses, denoted by a, b and c, then when A(a or b or c) is in an instance, then it is removed.

The empirical tests run through the simulator demonstrate that this One Difference Minimization extension does improve ASOCS ability to handle training-set-style data.

4.4.2 Priority ASOCSPriority ASOCS

This One Difference Minimization extension does not reduce the large number of instances generated by DVA when Non-Mutual

Exclusion is used. A solution to this dilemma is to use the Priority ASOCS model [Hugh89] rather than the ASOCS using DVA model. Recall from chapter 2 that this gives the NI priority to the OI without having to perform DVA, thus helping to maintain parsimony. The initial simulator does not perform Priority ASOCS mappings or store their instance sets.

A new simulator using Priority ASOCS is introduced. The previous topics of Mutual Exclusion, and One Difference Minimization are retested (figure 4.9). The tests show some interesting results:

1. Non Mutual Exclusion runs process faster and are just as reliable as the Mutually Exclusive sample sets.

2. The Non-Mutual sample sets finish in a short finite time, whereas with the basic ASOCS using DVA model, they do not finish at all.

Figure 4.9 - Priority ASOCS Test Results

	Priority ASOCS	Mutual Exclusion with Priority ASOCS					
Data Set	Compute Time	# Instances Matched	# Learned Instances	Compute Time	# Learned Instances	# Instances Matched Variables	Number
Zener Diode	1 minute	36/51	45/68	2.6 min	43/68	36/51 12	

Vote	37 min	6/47	301/388	52 min	301/388	6/47
						16
Hepatitis	6.45 min	0/16	137/139	11 min	137/139	0/16
						19

These results are a dramatic improvement from the results obtained with the basic ASOCS model, indicating that this is a step in the right direction for non boolean data.

4.4.3 Equal PriorityEqual Priority

Priority ASOCS, like the classical ASOCS model, uses an incremental learning process. Each new instance has a priority greater than any other old instance. In an attempt to make the algorithm behave in a more non-incremental fashion, equal priority (see section 2.6.1.3) is tested. One of the effects of this method is to make the highest priority much lower than the number of learned instances. ASOCS theory states that instance set fulfillment with this algorithm is identical to normal Priority ASOCS [Hugh89]p43. The results from the simulator runs using the last six data sets concur with the statement in [Hugh89].

Chart 4.10 shows test runs using these methods. Some general observations of the results include:

1. Using equal priority with or without Mutual Exclusion has the same recognition results as normal priority. This suggests that Priority ASOCS is consistent using this technique.

2. The learned instance set is smaller for equal priority than for normal priority. One possible reason is when instances have the same priority, normal and One Difference minimization combine or delete more instances. Another reason may be that the Contradictory Greater rule (section 2.6.1.2) does not occur as often, allowing more instances to be deleted.

Data Sets (All run on Priority Model)	# Learned Instances	# Matched Instances
Diode Mutual Binary Output	52/68	36/51
Diode Mutual Binary Output Equal Priority	32/68	36/51
Diode Mutual Input & Output	51/68	36/51 1 bad
Diode Mutual Input & Output Equal Priority	32/68	36/51 1 bad
Diode Non Mutual Input & Output	43/68	36/51
Diode Non Mutual Input & Output Equal Priority	43/68	36/51
Voting, Mutual	345/388	6/47
Voting, Mutual, Equal Priority	249/388	6/47
Voting, Non Mutual	301/388	6/47
Voting, Non Mutual, Equal Priority	301/388	6/47
Hepatitis, no Mutual	136/139	0/16
Hepatitis, non Mutual, Equal Priority	132/139	0/16
Hepatitis, Mutual	136/139	0/16
Hepatitis, Mutual, Equal Priority	132/139	0/16

Figure 4.10 Equal vs. Normal Priority in Priority ASOCS

4.5 <u>Missing Variables</u>4.5 Missing Variables

Once a better model is found, the research returns to improving the recognition rates. The Hepatitis and Voting data sets have many missing values in their training sets, a typical problem with training set data. Humans appear to deal with this by learning to filter out the errant values and guessing the correct answer. This is based upon previous experience and the context of the situation. The ASOCS model is not so robust. All variables in an instance are considered of equal significance. However, any variable with a wrong

or missing value is of less worth. Several approaches are discussed.

4.5.1 Reject Noisy InstancesReject Noisy Instances

The simplest solution is to reject any instance with a missing value, and request that it be resubmitted when all of the data is known. In the Hepatitis data set, there are many variables which are sporadically reported. This could be due to sloppy data gathering techniques. It would not be unreasonable to reject the bad instances and have them resubmitted when all the data is known. In the Vote data set, all variables are accurately reported. The don't know values may have several meanings and may be considered of less worth to a yes or no vote. In the case where the don't know is of equal validity to a yes or a no there would be no need for a resubmission of the instance. Rejection and resubmission are especially valid when the data set is large. The loss of noisy instances is minute when there are many others to learn from.

The Hepatitis data set is tested using this technique of rejecting noisy instances. There are 155 instances in total, of which 139 are used for training and 16 for testing. There are none recognized. (the topic of generalization will address this later). With the 75 bad instances (with one or more missing values) removed, there are 72 used for learning and 8 for testing. The result is still none recognized.

This approach is simple, and may be practical for many situations. The extension of rejection and resubmission of noisy instances suggests that for large data sets with noisy instances, this extension should make the training set more accurate and improve the recognition rate. Where the training set is small, and there is no

possibility to resubmit more accurate data, other solutions should be used.

4.5.2 Don't Care VariablesDon't Care Variables

ASOCS assumes that all variables are given accurate values, even though they may be inconsistent. 'If-then' rules in ASOCS are dealt with by omitting the Don't Care variables from each instance. This approach is also possible with training-set-style data. Training-set-style data would have any variable with an unknown value removed from the instance. There must be at least one variable remaining for the instance to be presented. Preferably most of the variables are accurate, otherwise the instance will contain little information. The remove of noisy variables implies that the training set data starts out with all instances having the same number of variables, but during learning, the number of variables per instance varies.

In figure 4.11 the last eight rows show the results of applying this technique to the voting and hepatitis data sets. The instance is completely rejected if the number of missing variables exceeds four. The limit of four removed variables helps keep the more noisy samples out of the test set. The *priority* column show whether normal or equal priority is used. The *1 Diff* column states whether 1 Difference Minimization is used. Note that the voting data has a slightly higher recognition rate. The higher recognition rate is due to some instances having fewer variables. The fewer variables result in superset or subset minimizations during learning, and superset recognitions during execution.

The hepatitis data still has poor recognition because of the

large number of variables with large numbers of discretization.

Training Set with Mutual Exclusion Input Variables using Missing Algorithm	Priority	1 Diff	# Learned Instances	# Instances Matched
Voting	Normal	Y	350/388	6/47
Voting	Equal	Y	282/388	6/47
Hepatitis	Normal	Y	138/139	0/16
Hepatitis	Equal	Y	139/139	0/16
Voting Don't Care	Equal	Y	250/388	8/47
Voting Don't Care	Normal	Y	320/388	8/47
Voting Don't Care	Equal	N	230/388	8/47
Voting Don't Care	Normal	N	267/388	8/47
Hepatitis Don't Care	Equal	Y	121/139	1/16
Hepatitis Don't Care	Normal	Y	124/139	1/16

Hepatitis Don't Care	Equal	N	122/139
			1/16
Hepatitis Don't Care	Normal	N	124/139
			1/16

Figure 4.11 Missing Data Tests

4.5.3 Differing Priorities of VariablesDiffering Priorities of Variables

Each missing or incorrect value in each instance could be given a lower priority. Prioritizing variables is complex to do, because the errant value could be on any variable. Enforcing different priorities per variable is difficult in the ASOCS model. Special handling and storage of different priorities in the learning and execution modes would significantly slow down the algorithms, especially for instances with large numbers of variables. Differing priorities per variable per instance is not a recommended approach.

4.5.4 Separate SubclassSeparate Subclass

Assign the missing variables to a separate subclass. The separate subclass would require no special processing. It will make generalization and learning more complex because there will be a duplication of instances: those with the variable known, and those with it unknown. This approach has the advantage of segregating out the missing values so that they do not hinder the known ones. However, the duplication of learned instances should increase the ASOCS network size. This increased amount of data might provide more information to the ASOCS model. Any noisy instance presented during execution will have a difficult time finding a

match because its noisy variables will be in separate subclasses from the known values. The number of instances with variables in these separate subclasses should be small, unless the data set is very noisy. If the noisy instance does find a match, the mapping will be suspect, because the missing variables may contain important classification information. e.g. A:? B:c D:? Z has two errant variables. Trying to find a match with other instances will be difficult because variables A and D are of an unknown value.

4.5.5 Separate Subclass with Special Considerations

Separate Subclass with Special Considerations

Assign the missing variables to a separate subclass with special processing. There are several approaches to the special processing:

4.5.5.1 Assume match always. This approach would apply to both learning and execution phases. Always matching would produce somewhat the same effect as eliminating the variable from the instance, except that the instances are treated as equal instead of subset, or superset. Always matching is not a good option as missing variable instances would receive higher, not lower priority. During execution there may be contradictory outputs because there would be more instances matched than really ought to be.

4.5.5.2 Assume Match Special Consideration. Give the missing values special consideration during learning. This option deals with missing values in a fairer manner. The first four lines of figure 4.11 show the results of this technique. Note that the number of instances recognized during execution phase is the same. The number of instances in the learned instance set are larger. The larger count is due to the fewer One Difference Minimizations performed. Remember that this extension will increase the complexity of an ASOCS implementation.

4.5.5.2.1 Consistency. Consistency checking has Equal discordant instances deleting the instance with the most missing variables, not just the oldest. DVA on Superset and Overlap (which rarely occur) are unchanged. This extension has the effect of discriminating against noisy instances in favor of more accurate ones. For example, if A:? B:b C:d D:a   Z is an OI and A:d B:? C:? D:a   Z is the NI, then the NI is deleted, not the OI, because it has more missing values. For Priority ASOCS with equal priority enabled (section 2.6.1.1.2), superset consistency keeps the NI and OI if the OI has fewer missing variables.

4.5.5.2.2 Minimization. Minimization with equal concordant instances removes the instance with the most missing variables. Subset and Superset minimization deletes the longer instance only if it has as many or more missing variables. This extension places missing variable instances in a lower priority, causing the instances to be removed sooner from the learned instance set.

4.5.5.2.3 One Difference Minimization. One Difference minimization which removes the discordant variable is not done if either of the instances has any missing variables. Otherwise, the one difference may be misleading and there may actually be more discordant variables. The objective is to maintain as accurate a mapping as possible. For example, if A:g B:e C:a   Z tries to minimize with A:? B:e C:a   Z, no action is taken. Remember that minimization is not necessary to keep a correct mapping. This algorithm is tried on the voting and hepatitis data sets. Figure 4.11 shows that no One Diff minimization occurs at all, because all instances had more than one variable different.

4.5.5.2.4 Execution. If all missing values are treated as matches, then the possibility exists that conflicting outputs (an error condition) could occur, as mentioned in 5.5.5.1. If the input state is A:g B:a D:b, then A:? B:a D:b   Z and A:g B:a D:b   Z match correctly, which yields a contradiction. Figure 4.11 suggests that there is less minimization with this algorithm, but that there are no incorrect matches during the execution phase. These training sets are too small to verify whether the execution phase produces inconsistent results.

All of the extensions to the basic ASOCS model discussed thus far have helped somewhat in improving ASOCS' ability to recognize training set data.

In conclusion, missing values are standard fare for training set data. They must be dealt with in some manner so as to generalize correctly.

4.6 Generalization4.6 Generalization

The ASOCS model maps discrete inputs to discrete outputs. This thesis does not discuss the internal architecture of how this could be done. There must be an exact match to some learned instance in the training set (which may have been minimized to produce more general instances) for recognition to take place. The ASOCS model has an indeterminate result if there is no match. In the real world, we rarely see an exact match, but settle for a closest choice.

Figure 4.12

Column explanation:
Test #: Which test run this is. Used for reference purposes.
1 Diff: Is One Difference Minimization used or not.
Mutual Excl Input: Is the input mapped to separate variables or is Mutual Exclusion used on the input variables.
Mutual Excl Output: Mutual Exclusion used on output variable.

Priority: There are three types of priority learning:
1. I means Non-Incremental method used,
2. E means Equal method used,
3. N means Normal ASOCS method used.

Learned: What is the final learned instance set count used during the execution phase.
% TTL Correct: During execution phase, what percentage of the test instances are correctly recognized.
Correct TTL: During execution phase, how many instances are recognized out of the total tested.
Correct Exact Recognize: How many instances which are matched with a complete and exact match of all input variables are recognized correctly.
Correct Close Recognize: How many instances which are matched with an incomplete (i.e. not all input variables match) match are recognized correctly.
Bad Exact Recognize: How many instances which are recognized with a complete match of input variables are recognized incorrectly (i.e. The learned instance set yielded an instance which has the wrong subclass versus the test instance).
Bad Close Recognize: How many instances which are recognized with an incomplete (i.e. not all input variables match) match are recognized incorrectly.

Test #1 can be described as using 1 Difference Minimization, with Mutually Exclusive input, and non-Mutually Exclusive output, Non-Incremental priority, 33 out of 68 training examples in the final learned instance set, 98% correct matching on the test examples (50/51), 14/15 close matches, no bad exact matches, and one bad close match.

Diode Data

Test #	1 Diff	Mutual Excl Input	Mutual Excl Output	Priority (Normal ,Equal, nonIncremental	# Learned	% Correct	TTL Correct TTL	Correct Exact Match	Correct Close Recognize	Bad Exact Match Bad Close Match
1	Y	Y	N	I	33/68	98	50/51	36/36	14/15	0/36 1/15
2	Y	Y	N	E	32/68	98	50/51	36/36	14/15	0/36 1/15

3	Y	Y	N	N	53/68	98	50/51	36/36	14/15	0/36 1/15
4	N	Y	N	I	43/68	98	50/51	36/36	14/15	0/36 1/15
5	N	Y	N	E	43/68	98	50/51	36/36	14/15	0/36 1/15
6	N	Y	N	N	43/68	98	50/51	36/36	14/15	0/36 1/15
7	Y	Y	Y	I	34/68	90	46/51	35/36	11/15	1/36 4/15
8	Y	Y	Y	E	32/68	86	44/51	35/36	9/15	1/36 6/15
9	Y	Y	Y	N	51/68	86	44/51	35/36	9/15	1/36 6/15
10	N	Y	Y	I	43/68	88	45/51	35/36	10/15	1/36 5/15
11	N	Y	Y	E	43/68	86	44/51	35/36	9/15	1/36 6/15
12	N	Y	Y	N	43/68	86	44/51	35/36	9/15	1/36 6/15
13	Y	N	N	I	45/68	96	49/51	36/36	13/15	0/36 2/15
14	Y	N	N	E	43/68	98	50/51	36/36	14/15	0/36 1/15
15	Y	N	N	N	43/68	98	50/51	36/36	14/15	0/36 1/15
16	N	N	N	I	45/68	96	49/51	36/36	13/15	0/36 2/15
17	N	N	N	E	43/68	98	50/51	36/36	14/15	0/36 1/15
18	N	N	N	N	43/68	98	50/51	36/36	14/15	0/36 1/15
19	Y	N	Y	I	44/68	88	45/51	35/36	10/15	1/36 5/15
20	Y	N	Y	E	43/68	86	44/51	35/36	9/15	1/36 6/15
21	Y	N	Y	N	43/68	86	44/51	35/36	9/15	1/36 6/15
22	N	N	Y	I	44/68	88	45/51	35/36	10/15	1/36 5/15
23	N	N	Y	E	43/68	88	45/51	35/36	9/15	1/36 6/15
24	N	N	Y	N	43/68	88	45/51	35/36	9/15	1/36 6/15

Figure 4.13

For example, test #25 is interpreted as One Difference minimization used, Mutually Exclusive input, Non-Incremental priority used, with 260/388 training examples in the final learned instance set, 85% correct matches (40/47), 6 exact matches, 34 correct close matches, and 7 bad matches.

Voting Data

Test #	1 Diff	Mutual Excl Input	Priority (Normal, Equal, nonIncremental)	# Learned	% Correct	TTL Correct Total	Correct Exact Recognize	Correct Close Recognize	Bad Close Recognize
25	Y	Y	I	260/388	85	40/47	6	34	7
26	Y	Y	E	249	85	40	6	34	7
27	Y	Y	N	345	83	39	6	33	8
28	N	Y	I	301	85	40	6	34	7
29	N	Y	E	301	85	40	6	34	7
30	N	Y	N	301	83	39	6	33	8
31	Y	N	I	301	85	40	6	34	7
32	Y	N	E	301	83	39	6	33	8
33	Y	N	N	301	83	39	6	33	8
34	N	N	I	301	85	40	6	34	7
35	N	N	E	301	83	39	6	33	8
36	N	N	N	301	83	39	6	33	8

Figure 4.14

This table has one extra column to show the maximum number of missing values tolerated before an example is completely rejected.

Voting Data using Don't Care for Missing Data, all Mutual Exclusion

Test #	1 Diff	# Variables Missing	Priority (Normal, Equal, nonIncremental	# Learned	% Correct	TTL Correct Total	Correct Exact Recognize	Correct Close Recognize Bad Close Recognize
85	Y	<3	I	251	85	40	8	32 7
86	Y	<3	E	239	85	40		32 7
87	Y	<3	N	307	83	39		31 8
88	N	<3	I	240	85	40		32 7
89	N	<3	E	229	85	40		32 7
90	N	<3	N	251	83	39		31 8
91	Y	<5	I	262	85	40		32 7
92	Y	<5	E	250	87	41		33 6
93	Y	<5	N	320	85	40		32 7
94	N	<5	I	243	85	40		32 7
95	N	<5	E	230	85	40		32 7
96	N	<5	N	267	83	39		31 8

Figure 4.15

Test #37 can be interpreted as One Difference minimization is used, with Mutually Exclusive inputs, using Non-Incremental priority learning, with 128/139 training examples in the learned instance set, 75% correct matches (12/16), all of which are close matches, with 4 incorrect matches.

Hepatitis

Test #	1 Diff	Mutual Input	Priority (Normal ,Equal, nonIncremental	# Learned /139	% Correct	T T LCorrect Close Recognize Bad Close Recognize
37	Y	Y	I	128	75	12 4
38	Y	Y	E	129	75	12 4
39	Y	Y	N	136	75	12 4
40	N	Y	I	138	75	12 4
41	N	Y	E	138	75	12 4
42	N	Y	N	138	75	12 4
43	Y	N	I	132	69	11 5
44	Y	N	E	132	69	11 5
45	Y	N	N	136	69	11 5
46	N	N	I	137	69	11 5
47	N	N	E	137	69	11 5
48	N	N	N	137	69	11 5

Figure 4.16

Test #98 can be interpreted as One Difference minimization used, a maximum of 2 missing variables tolerated, Equal priority learning, 113 / 139 training examples in the learned instance set, 69% recognition rate, with no exact matches, 11 correct close matches, and 5 bad close matches.

Hepatitis using Don't Care algorithm, all Mutual Exclusion

Test #	1 Diff	# Variables Missing	Priority (Normal, Equal, nonIncremental)	# Learned /139	% Correct	TTL Correct Exact Recognize	Correct Close Recognize Bad Close Recognize
97	Y	<3	I	115	**69**	**0/16**	**115**
98	Y	<3	E	113			
99	Y	<3	N	119			
100	N	<3	I	120			
101	N	<3	E	120			
102	N	<3	N	120			
103	Y	<5	I	120		**1/16**	**10**
104	Y	<5	E	121			
105	Y	<5	N	124			
106	N	<5	I	122			
107	N	<5	E	122			
108	N	<5	N	124			

Figure 4.17

Test # 51 from the Monk 1 data set can be interpreted as One Difference minimization used, no Mutual Exclusion on input variables, Normal priority learning, 124/124 in the learned instance set, 75% correct recognition rate, 326 out of 432 correct matches, all the exact matches are correct, and 202 of the close matches are correct, with 106 bad close matches.

Monk

Test #,	1 Diff	Mutual Excl Input	Priority (Normal ,Equal, nonIncremental	# Learned	% Correct	TTL Correct Total	Correct Exact Recognize	Correct Close Recognize	Bad Exact Recognize Bad Close Recognize

Monk Data Set 1. 124 Learning Instances, 432 Test Instances									
49	Y	N	I	124	78	336	124	212	0 96
50	Y	N	E	124	75	326	124	202	0 106
51	Y	N	N	124	75	326	124	202	0 106
52	N	N	I	124	78	336	124	212	0 96
53	N	N	E	124	75	326	124	202	0 106
54	N	N	N	124	75	326	124	202	0 106
55	Y	Y	I	70	78	337	124	213	0 95
56	Y	Y	E	68	77	333	118	215	0 99
57	Y	Y	N	76	75	322	120	202	0 110
58	N	Y	I	124	78	339	124	215	0 93
59	N	Y	E	124	78	339	124	215	0 93
60	N	Y	N	124	75	326	124	202	0 106
Monk Data Set 2. 169 Learning Instances, 432 Test Instances									
61	Y	N	I	169	64	277	169	108	0 155
62	Y	N	E	169	76	330	169	161	0 102
63	Y	N	N	169	76	330	169	161	0 102
64	N	N	I	169	64	277	169	108	0 155

65	N	N	E	169	76	330	169	161	0 102
66	N	N	N	169	76	330	169	161	0 102
67	Y	Y	I	91	64	276	169	105	0 158
68	Y	Y	E	95	63	273	163	110	0 159
69	Y	Y	N	131	75	326	166	160	0 106
70	N	Y	I	169	64	277	169	108	0 155
71	N	Y	E	169	64	277	169	108	0 155
72	N	Y	N	169	76	330	169	161	0 102
Monk Data Set 3. 122 Learning Instances, 432 Testing Instances									
73	Y	N	I	122	82	354	116	238	6 72
74	Y	N	E	122	81	352	116	236	6 74
75	Y	N	N	122	81	352	116	236	6 74
76	N	N	I	122	82	354	116	238	6 72
77	N	N	E	122	81	352	116	236	6 74
78	N	N	N	122	81	352	116	236	6 74
79	Y	Y	I	69	83	357	116	241	6 69
80	Y	Y	E	71	82	354	113	241	6 72
81	Y	Y	N	84	82	355	114	241	6 71
82	N	Y	I	122	82	354	116	238	6 72
83	N	Y	E	122	82	354	116	238	6 72
83 Mix	N	Y	E	122	57	248	116	132	6 178
84 Mix	N	Y	N	122	80	344	116	228	6 82

84	N	Y	N	122	81	352	116	236	6 74

The large number of variables in the last six data sets make exact matches difficult. In the hepatitis data set, the chances of all 19 variables, each discretized four ways, matching another instance are 1 / 4^{19} or 1 in 2.74878 $*10^{11}$. Clearly, many close matches go undetected. In all test runs for the hepatitis data set, there are no recognitions. The Vote data set has only 6 out of 47 test instances recognized with a training set of 388. The best runs on the Zener Diode set yield 36 out of 51 when a discretization level of 4 is used.[1]

4.6.1 Hamming DistanceHamming Distance

The extension of accepting close matches, or short Hamming distances is introduced. With this extension, the results improve dramatically. The *Hamming Distance between two instances* is defined as:

1. For Mutually Exclusive Variables: The number of variables which have different subclasses plus the number of variables not found in both instances. For example,

A:a B:c C:b Z and A:b B:c C:b Z have a Hamming distance of 1 because A:a and A:b are discriminant.

2. For Non Mutually Exclusive Variables: The number of variables not found in both instances plus the number of variables with different polarity. e.g. A B C D Z and

A C D E Z have a Hamming distance of 2 because B and E are

[1] In Lloyd Allred's research, that is the best rate he got on 100% certainty of classification. He was able to predict almost all the rest with a level of uncertainty [Allr88].

overlap variables.

If no perfect match is outputted when the test instance is presented to the learned instance set, a list is kept of the outputs with the closest Hamming distances (least number of incorrect variable matches). The model outputs the list of closest matches, along with the Hamming distance. The Hamming distance is an indicator of the reliability of the answer. Exact matching has a Hamming distance of 0. When there are multiple instances selected, the one(s) with the highest priority determine the output polarity and subclass. There are usually several instances selected when equal priority learning is used. These instances may have contradictory output values. An additional count is made of which output polarity and subclass occurs the most. This extra count helps resolve the vacillation deadlock of contradictory outputs.

4.6.2 Non-Incremental AlgorithmNon-Incremental Algorithm

At this juncture a further attempt is made to provide a non-incremental learning algorithm for Priority ASOCS. Because consistency and minimization are done as each NI is presented, the order of presentation can affect which instances are minimized or deleted out of the learning set [1]. Therefore, in order to give all instances an equal opportunity, no minimization or consistency should be done until all instances are present. In reality, all instances are of equal value, therefore all should be given equal priority. For the test runs used in this thesis, a priority of 1 is

---

[1] Test runs 83 and 84 have their order of learning instances modified deliberately to show that there is decreased recognition of the close instances. The changes are done so that more incorrect recognitions occur. These test runs are called 83 Mix and 84 Mix.

chosen. This algorithm is presented and tested as an avenue for discovery, not as a conclusive proof for a theorem.

The issue of consistency may not be more important than minimization. For our purposes consistency supersedes minimization in the learning algorithms. The non-Incremental algorithm has these modifications to the learning algorithm:

1. The order of presentation affects which instances are deleted if there are inconsistent equal instances. To avoid this problem, all inconsistent instances are stored in a separate list.
2. Every time an instance is duplicated, its priority is incremented.
3. When all instances have been presented once, the inconsistent set is presented again. The instance with the highest priority (i.e. more frequent occurrence) remains and the other is deleted. In the case of a tie, the deletion is arbitrary. This process implicitly gives higher priority to the more common instances.
4. Finally, all instances are presented to all others for minimization. Here the order of presentation has some affect on which minimizations occur. Recall from chapter three that minimizations are not unique. This minimization affects its parsimony and generalization (learning).

This algorithm brings up a couple of issues:

1. Each instance is presented at least twice, and as many as 3 times, rather than just once. The first presentation does no minimization, to avoid any order bias. The second iteration is to resolve inconsistency, after all instances have been presented. Finally, after the instance set is minimized. Note, the first two presentations do no minimization. This allows consistency to not be biased by

minimizations which are affected by the order of presentation. This algorithm would be slightly slower during the learning process.

2. This process gives each instance equal opportunity, and removes the problem of inconsistent instances being prematurely deleted. Only after all instances have been compared are the most infrequently occurring inconsistent instances deleted.

With these techniques, the recognition rates rise dramatically for all of the last six data sets (see Figure 4.18). Figures 4.12 through 4.17 give a complete list of the six training sets and their results with generalization by closest Hamming distance, using the three types of priority learning (Normal, Equal, Non-Incremental).

Zener Diode	**36 + 14**	**= 50/51**	**98%**
Vote	**6 + 34**	**= 40/47**	**85%**
Hepatitis	**0 + 12**	**= 12/16**	**75%**
Monk 1	**124 + 212**	**= 336/432**	**78%**
Monk 2	**169 + 161**	**= 330/432**	**76%**
Monk 3	**116 + 238**	**= 354/432**	**82%**

Figure 4.18 - Exact and Close Recognition Improvements

In all six data sets, the generalization (close) matches become random when the Hamming distance is greater than three. In other words, the probability of a successful match rises as the Hamming distance decreases (i.e. more variables are matched).

The empirical tests done in this thesis suggesst that the extensions to the basic ASOCS model, discussed in this thesis, yield significant improvements in the number of correct matches on training set data.

4.6.3 Analysis of Results Analysis of Results

The six extensions discussed in this thesis are:

1. Mutual Exclusion for Variables requiring more than two subclasses.

2. Missing Variable learning algorithms for noisy data.

3. Priority ASOCS model.

4. Equal Priority for learning.

5. Non-Incremental Learning.

6. Hamming Distance Recognition Metric.

This section evaluates how these extensions to the basic ASOCS model improve the recognition rates for the last six data sets.

4.6.3.1 <u>Diode Training Set.</u> (figure 4.12) Mutually Exclusive output yields the poorest results. These poor results are due to difficulty in classifying some of the bad diode types. However, the recognition between good and bad diodes is excellent. The largest learned instance sets occur with normal priority. The larger instance size is due to the differing priorities triggering the Contradictory Greater situation more frequently, thus prohibiting some minimization. The smallest learned sets are found when One Difference minimization and Mutual Exclusion on input variables are used. The close recognition helps substantially in obtaining almost 100% correct recognition rates.

4.6.3.2 <u>Voting Training Set.</u> (figures 4.13, 4.14) All results are very closely grouped, with 6/47 exactly recognized and 33 to 34 out of 47 close recognitions in all cases except for the Missing Variable algorithm. Here too, the smallest learned instance sets use Equal or Non-Incremental learning, and One Difference Minimization and

Mutual Exclusion on input variables. Without close recognition, the matching rate is 13%, with the Hamming distance metric the recognition rate is 85%. This generalization technique appears to radically improve ASOCS performance.

The Missing data algorithms (figure 4.14) show the same recognition rates, even though the number of learned instances is less. This consistent recognition rate may suggest that the noisy instances which are not included provide little information.

4.6.3.3 <u>Hepatitis Training Set.</u> (figures 4.15, 4.16) There are no exact recognitions except in the Missing data algorithm. The poor recognition rate demonstrates that noisy and errant data can be difficult to recognize. With the Hamming distance technique the results are much better (69 -75 % recognition). Because of the small training and test set sizes, the one less instance recognized in Non-Mutual Exclusion causes the large swing in results. The reason the Non-Mutually Exclusive tests faired poorer is because of overlap recognition. In Mutual Exclusion, all variables are recognized, making for no overlaps in consistency or minimization. Rather, there are some discriminant variables found between all instances. With Non-Mutual Exclusion, all variables are mapped to several subclasses by using completely unrelated values (a b c d for 4 subclasses). Instances which have several overlap variables between each other have difficulty discriminating correctly. This difficulty is an important point to recognize about the usefulness of Mutual Exclusion in training set data.

The Missing data algorithms (figure 4.16) perform as well here as in the voting data, also suggesting that the noisy instances

provide little recognition information.

4.6.3.4 <u>Monk Training Sets.</u> (figure 4.17) The three rules used to produce the test sets yield somewhat different results. As expected, set 3 with some error introduced produced the best results (83 - 81 %)[1]. All three sets display the same phenomenon as described in the hepatitis training set. Note that the Equal priority tests do either poorer (Set 1 & 3) or better (set 2) when Mutual Exclusion is not used. This difference in performance is due to overlap variables not providing a discriminant variable, so higher priorities have to be used. With Mutual Exclusion, there is always at least one discriminant variable, and usually 2 or more. This discriminant variable leaves the priority at 1, so that the instances recognized during execution phase are not different in their priority. When contrasted with the Non-Mutually Exclusive tests, the order of presentation affects the results. This anomaly does not occur in Non-Incremental and Normal learning tests. In Non-Incremental learning, the order does not affect the priority. In Normal priority tests, the priority follows the order, regardless of Mutual Exclusion. Note that the differences in recognition rates occur in the close, not exact categories. Also, note that when One Difference minimization and Mutual Exclusion are combined, the learned instance set is much smaller due to increased minimization.

The Monk data has no inconsistencies in its learning in groups 1 and 2. Monk 3 has inconsistencies because errors are introduced.

---

[1] Sets one and two are encoder problems, requiring more complete training sets. Set 3 is more of an and/or problem. For set 3 a smaller training set is able to provide enough information to generalize.

The Monk data sets are different from the other training sets because they are generated from precise functions (see section 3.6). The other training sets are a collection of data from "real world" sources. However, the Monk data does provide a good basis from which to study some of ASOCS's fundamental characteristics when extensions are implemented.

In conclusion, five problems to the basics ASOCS instance set model are presented. Solutions to four of these problems are suggested.

5. Conclusions and Future Research5. Conclusions and Future Research

This chapter is a summary of the problems found using training-set-style data in the ASOCS neural network, and the extensions used to address some of these problems.

5.1 Theorems and Proofs5.1 Theorems and Proofs

The extensions discovered in this thesis have not been rigourously proven. The extensions have been discussed, implemented into simulators and tested using a limited set of data. Many of the extensions show promise for the basic ASOCS model to handle training-set-style data. Future research could be done to provide theories and proofs for the work done in this thesis.

5.2 Mutual Exclusion5.2 Mutual Exclusion

The training-set-style data used to study the ASOCS model has many variables. Many of these variables need to be subclassified (discretized) into more than 2 subclasses. These multiple subclasses generate a problem of large learned instance set growth due to Discriminant Variable Addition when separate variables are used to denote the subclasses. Learning times become intractable. For example, if the OI

A F T Y   Z and the NI B C D J   Z are to be consistent, then DVA causes the OI to become 4 modified instances (there are no discriminant variables).

This problem can be resolved by using a major class for the variable and a minor subclass for the discretization. These subclasses are mutually exclusive. For example, A:a B:d C:f D:b   Z and A:b B:c C:d D:a   Z have several discriminant variables so that DVA is not necessary. DVA and learning times are reduced.

5.3 Arbitrary Discretization

5.3 Arbitrary Discretization

The mapping of training-set-style data to ASOCS instances is done by arbitrarily mapping the variables into evenly discretized subclasses. This evenly distributed mapping results in poor subclassification due to subclass boundaries breaking up natural groupings of the data. Although no solution is presented in this thesis, discussion is made of its importance and current work being done in this area of research. This topic is an excellent avenue for further research and study.

5.4 One Difference Minimization

5.4 One Difference Minimization

The use of Mutual Exclusion with One Difference Minimization causes a matching problem when using the algorithm discussed in section 2.5.2.2. For example,

A:a B:b   Z and A:c B:b   Z yields A:(a   c) B:b   Z. The A variable is not completely removed, rather its subclasses are *ORed* together. This is a superset instance when compared to the two instances which generate it, even though variable A is not removed. This matching problem requires a modification to the learning algorithm such that superset comparisons are used during learning, while the execution stage treats the two superset instances as being recognized. This extension to the basic ASOCS model permits One Difference Minimization to be properly utilized when Mutual Exclusion is used for discretization.

This topic needs further research with large data sets to verify that variables do minimize into one instance with their entire range *ored* together.

5.5 Priority ASOCS5.5 Priority ASOCS

Discriminant Variable Addition is a problem when Non-Mutual Exclusion is used. The Priority ASOCS model is a solution to this inefficiency. The priority of each instance is based upon the order of presentation of the instances. DVA is eliminated and learning times are shorter.

5.6 Incremental Learning5.6 Incremental Learning

Both the basic ASOCS with DVA and the Priority ASOCS model are based upon an incremental learning model. This learning model is contrary to the style of learning which training set data require. Training set examples need to be treated as equals, with no preference based upon order of presentation. Two solutions are discussed in chapter 4:

5.6.1 Equal PriorityEqual Priority

This solution has the priority on new instances incremented only when an inconsistency is resolved during learning. Thus the priority shows inconsistency. This modification is still an incremental style learning algorithm, but it has the appearance of a non-incremental learning algorithm when Mutual Exclusion is used. This appearance is due to the continual presence of discriminant variables, which resolve inconsistencies.

5.6.2 Non-Incremental PriorityNon-Incremental Priority

This algorithm treats all learned instances equally, and does not eliminate any inconsistent instances until all instances have been presented. Duplicate instances are given higher priority. No minimization is done until the instance set is consistent. This algorithm gives fairer treatment to training-set-style data, but still assumes that consistency is more important than minimization. Noise and errors in training set data yield inconsistencies which must be reduced by generalization.

Further research needs to be done with non-incremental learning where minimization is given higher precedence versus consistency. Hopefully, this higher precedence would allow generalization to reduce training set noise.

5.7 Missing Data5.7 Missing Data

Some training-set-style data is incomplete, with some variables containing erroneous or non-existent values. Three approaches to this problem are discussed:

5.7.1 Reject Noisy ExamplesReject Noisy Examples

Any instance with a missing value is rejected and then resubmitted when complete. This algorithm is only viable if the data can be resubmitted or the data set is large enough to tolerate the lost knowledge.

5.7.2 Don't CareDon't Care

Treat the missing variables as *Don't Care* by removing them from the instances. This removal is done up to a prescribed limit, after which the instance is removed. This algorithm shows slightly smaller learned instance sets due to more superset minimization and rejected instances, and equivalent recognition rates to the other missing data algorithms.

5.7.3 Modify Learning AlgorithmModify Learning Algorithm

Missing data are considered matches with special consideration. Partial examples have lower priority during consistency, and minimization of complete (no missing values) instances with partial instances is prohibited. This algorithm results in larger learned instance sets due to less minimization, but the test sets are too small to verify improved recognition rates.

This topic warrants further study with large training sets of data. Hopefully, some of the anticipated results (improved recognition rates, learned instance set sizes) would manifest themselves.

5.8 Generalization5.8 Generalization

The ASOCS model only recognizes instances which completely and exactly match an instance in the learned instance set. In order to recognize instances which are close, i.e. not exact matches, a Hamming distance metric is defined. The Hamming distance is the number of variables between instances which have different subclasses, or which are nonexistent in both instances. This extension radically improves recognition rates for all of the last six training sets.

Further research needs to be done with this generalization metric and other distance schemes to find ways of improving recognition of instances not exactly matched to the learned instance set.

5.9 Large Test Sets5.9 Large Test Sets

All test sets used in this thesis have under 400 training examples and under 450 test instances. The hepatitis training set has only 16 test instances. These extensions to the basic ASOCS model need to be tested using much larger sets of data. Some of the anticipated results have not manifested themselves clearly, or at all, because the data sets are too small. These anticipated results include:

1. Missing data algorithm recognition rates,
2. Non-Incremental learning algorithm recognition rates,
3. One Difference minimization and removal of complete range variables.

Further research using large data sets should help in better understanding the ramifications of these extensions and provide better evidence for their refinement. Also, larger trainings sets should help develop new extensions.

In conclusion, the extensions to the basic ASOCS model, presented in this thesis, tend to show improvement in the ability of the ASOCS model to handle training-set-style data that is processed by other neural networks.

6. Appendix: Simulator6. Appendix: Simulator

The simulator was written in TurboC 2.0 for the IBM PC, and may be ported over to the MacIntosh using Light Speed C. A substantial amount of work has gone into its development. Source code is available from Edward F. Hart. Its purpose is twofold:

1. To provide a means of verifying and testing the ASOCS extensions and problems for this thesis. The raw data is mapped into ASOCS instance sets by small programs, and then run through the simulator;

2. To allow others access to a tool to learn and study the ASOCS neural network model. This simulator is not the purpose of this thesis, just a means to accomplish it.

The internal storage of the instance sets is done by dynamically allocated link lists. This facilitates efficient memory usage when large sets of instances are learned. There can be up to 9 output variables and 200 input variables per instance. One thousand instances is the programmed limit, but simulations with up to six thousand instances, each with fifteen input variables have been done. The manipulation of these linked lists in serial fashion is very memory intensive. Fast processors radically reduce the learning and execution times.

The screen is as follows:

```
…õõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõ A S O C S õõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõª
∫ Detail ON    Screen ON    Hard copy ON ∫ Instance #:3        Total #:3          ∫
∫ Input  FILE LEARN/Execute     sTep ON ∫ Ttl Time:1806       Consist:2          ∫
∫ One diff ON        one difF subset     ∫ Min:                                   ∫
∫     Add:1       Del:         Mod:      ∫Ttl Add:3       Del:          Mod:      ∫
∫Contradict  Minimize      One Diff      ∫Contradict  Minimize      One Diff      ∫
∫Sub :        Sub  :        Sub  :       ∫Sub  :        Sub  :        Sub  :      ∫
∫Eq   :       Eq   :        Eq   :       ∫Eq   :        Eq   :        Eq   :      ∫
```

```
∫Supr:         Supr:          Supr:         ∫Supr:         Supr:          Supr:          ∫
∫Over:                        Iter:         ∫Over:                        Iter:          ∫
∫NIA:b B:c C:b D:b E:c F:c G:d H:a I:b J:b K:d L:d  > a:c                                ∫
Ãõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõõπ
∫CR to Continue                                                                          ∫
∫                                                                                        ∫
∫                                                                                        ∫
» 1 Show Set 2 Consistency 3 Delete 4 Modify G Go/Stopõõõõõõõõõõõõõõõõõõõõõõõõõõõ°
```

The program is initiated by typing 'ASOCS'. A default initialization file called ASOCS.CON is retrieved. If it doesn't exist, default values are assigned. Here is a sample:

Step On
One Diff On
Hard Copy On
Detail On
One Sub On
File In d4.run
Screen On

Step is On or Off, with On the default. It controls whether the program stops after each instance is presented, or keeps processing instances. For beginner students of ASOCS, the Step should be on.

One Diff is On or Off, with On the default. It controls the use of One Difference minimization during the learning of new instances. One Difference slows down processing speed, but does provide a significant portion of the minimization which occurs.

Hard Copy is On or Off, with Off the default. If it is on, the file ASOCS.OUT is built as the program reads in instances. If a ASOCS.OUT already exists, it is appended to, rather than erased. Every instance inputted during learning is shown. At the end of learning, summary statistics of the fields shown on the screen are listed, as well as a list of all final instances retained in the network. During execution phase, each instance presented is listed. If no matches are found, then the instance number and an arrow are on the next line, signifying that it wasn't learned. If matches are found, all of them are listed with an asterisk in the front of each. At the end of execution, the total number of instances presented for matching, and the number actually recognized are listed. After every twenty instances which are presented to the program, the output file is saved and reopened. This assists long, unattended program runs where a crash may occur.

Detail is On or Off, with On the default. This option stores copies of the modified or deleted instances whose total counts are listed on the screen. These instances can be listed when the program stops in Step mode. How to do this will be described later. Program execution is slower with this option, and is useful only when Step is also on. For large sets of instances, this option should not be used.

One Sub is On or Off, with On the default. One Difference Subset minimization can be very slow if the ripple effect (see the description of rebroadcasting in chapter three) causes multiple iterations of this process. The One Diff option must also be on for this option to work.

File In is followed by a file name. This file must contain instances only. There are two special symbols which tell the program to stop learning and start executing '#', and to stop '%'. A sample of this file could look like this:

```
A:b B:a C:b D:b E:b F:d G:d H:a I:a J:a K:a L:a  > a:f
A:b B:c C:a D:a E:a F:d G:d H:a I:a J:a K:d L:d  > a:a
A:b B:c C:b D:b E:b F:c G:d H:a I:b J:b K:d L:d  > a:a
A:b B:a C:a D:a E:a F:d G:d H:a I:a J:a K:a L:a  > a:f
#
A:b B:a C:b D:b E:b F:d G:d H:a I:a J:a K:a L:a  > a
A:c B:b C:b D:b E:c F:c G:d H:a I:b J:b K:c L:d  > a
```

```
A:b B:a C:b D:b E:b F:c G:d H:a I:a J:a K:a L:a  > a:f
A:d B:c C:b D:b E:c F:c G:d H:a I:b J:b K:c L:d  > a:e
%
```

Each instance variable on the input and output side must be separated by a space. The colon is used to denote the class versus subclass for a multi-state variable. Variables which are only positive or negative use the   to denote negative. Output variables in the execution mode are there to calculate correct recognitions. All of the simulation runs performed are done with instance sets prepared as files. Small programs are written to take raw data and convert it into discretized instance sets. The raw data (as exampled in chapter 4) is read in as ASCII strings to a parser which converts the analog or binary data into ASCII representations of ASOCS input variables, and an ASOCS output variable.

Screen is On or Off, with On the default. This displays the numbers on the screen for the various types of minimization and consistency, as well as the time. If you are going to run a large instance set unattended, then turn this off.

There are four command line parameters which select priority learning methods.

1. I means non-Incremental learning algorithm (see section 4.6.2).
2. D-# means Missing Data Don't Care algorithm with # of variables as a limit for the number missing values to tolerate before rejecting instances.
3. M Missing Data algorithm as described in section 4.5.5.2.
4. P Equal Priority algorithm.

All of these status types are displayed in the upper left corner of the screen. The times for total run, consistency and minimization are in the upper right corner. On the IBM PC they are in 18ths of a second. The Total # is the count of learned instances retained during learning mode. The Instance # shows how many instances have been presented so far. The top left window shows the number of contradictions, minimizations and one difference minimizations for the last instance. The detail numbers are for Subset, Equal, Superset and Overlap. The top right window shows the accumulated totals for each of the numbers.

Each new instance is shown in the middle of the screen at the top of the bottom window at the NI prompt. The bottom window shows lists of instances. These lists come from recognized instances during execution mode, or detail lists during learning mode.

If Step and Detail are on, then 4 for Modify will allow you to Tab around the screen to any upper left window status or count field. After you have tabbed to the correct field, a Carriage Return will activate that field. At the status fields, the status will be toggled from On to Off (or Off to On), or Learn to Execute, or Input from file to screen. At each of the count fields, if there are instances to list, they will show in the bottom window. If Hard Copy is on then these will also be stored on the disk. Notice that each of the status fields has one capital letter. Rather than tab around the screen, you may use the letter to toggle the corresponding field.

The other prompts on the bottom of the screen are self explanatory. 1 for Show Set will list the currently learned instance set. 2 for Consistency will check the correctness of the learning algorithms by running a self test on the instance set. It should report no errors. 3 for Delete will ask for what instance # to delete. Go/ Stop will only work when Step is off. While the program is running you may stop it and perform any of the four menu options. Then you can Go again.

If input is not from a file, then you will be prompted to enter in an instance.

With this simulator a student of ASOCS will see how:

1. minimization affects the instance set,
2. contradiction is handled with Discriminant Variable Addition,
3. Priority ASOCS improves performance,
4. the extensions discussed in this thesis affect recogntion rates.

It is hoped that users will find it a user friendly, instructive tool.

7. Bibliography7. Bibliography

Allr88 Allred, Lloyd G., "*Neural Networks in Automatic Testing of Diode Protection Circuits*," IEEE Autotestcon ' 88

Ackl85 Ackley, D. H., Hinton, G. E., and Sejnowski, T. J., "*A Learning Algorithm for Boltzmann Machines.*" Cognitive Science, 1985, Vol 9, pp 147-169.

Bert92 Bertelson, Rick. Thesis in progress at Brigham Young University under Dr. Tony Martinez. Researching unsupervised learning.

Carn91 S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S.E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang, "*The MONK's Problems - A Performance Comparison of Different Learning algorithms*" Technical Report CS-CMU-91-197, Carnegie Mellon University, December 1991. Sebastian Thrun editor.

Fors84 Forsyth, Richard, "*Expert Systems: Principles and Case Studies,*" University Press, Cambridge, 1984

Gong88 Gong, G. "*Hepatitis Domain,*" Carnegie-Mellon University via Bojan Cestnik, Jozef Stefan Institute, Jamova 39, 61000 Ljubljana, Yugoslavia.

Hopf85 Hopfield, J. J., Tank, D. W., " *'Neural' Computation of Decisions in Optimization Problems,*" Biological Cybernetics Vol 52, 141 - 152, 1985

Hopf86 Hopfield, John J., Tank, David W., "*Computing with Neural Circuits: A Model,*" Science, Vol 233, 8 Aug 1986, pp 625- 633

Hugh89 Hughes, Brent, "*Priority ASOCS Neural Network*", Thesis paper for Masters of Computer Science, Brigham Young University , 1989.

INNS88 International Neural Network Society Journal, Supplement, Abstracts from 1988 Conference. pp 227, 226, 447, 455, 460, 465, 477. Boston Pergamon Press.

John85 Johnson, L., Keravnou, E. T., "*Expert Systems Technology: A Guide,*" Abacus Press, 1985

Koho82 Kohonen, Teuvo, "*Self-Organized Formation of Topologically Correct Feature Maps*", Biological Cybernetics No 43, pp 59-69, 1982 , Springer Verlag.

Kosk87 Kosko, Bart, "*Competitive Adaptive Bidirectional Associative Memories,*" Int. Conf. on Neural Nets, San Diego, June 1987

Kosk88a Kosko, Bart, "*Adaptive Bidirectional Associative Memories,*" Applied Optics, Vol 26 No 23, pp 4947 -4960, December 1987

Kosk88b "*Bidirectional Associative Memories,*" IEEE Transactions on Systems, Man, and Cybernetics, Vol 18, No 1, pp 49 - 60, Jan 1988

Mart86 Martinez, Tony, "*Adaptive Self-Organizing Logic Networks,*" U of California, Los Angeles, 1986

Mart87 Martinez, Tony, "*Models of Parallel Adaptive Logic,*" Proceedings of the 1987 IEEE Systems Man and Cybernetics Conference, pp 290 - 296.

Mart88a Martinez, Tony, "*Digital Neural Networks,*" Proceedings of the IEEE Systems Man and Cybernetics Conference, pp 681- 684, 1988

Mart88b Martinez, Tony, Vidal, Jacques J., "*Adaptive Parallel Logic Networks,*" Journal of Parallel and Distributed Computing, Vol 5, No 1, pp 26-58, Feb 1988

McCl86 McClelland, Rumelhart, and PDP Research Group, "*Parallel Distributed Processing,*" Vol. 1 and 2, MIT Press, 1986.

Rich83 Rich, Elaine. "*Artificial Intelligence,*" McGraw-Hill, 1983

Schl87 Schlimmer, Jeff. "*1984 United States Congressional Voting Records Database,*" Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, Volume XL: Congressional Quarterly Inc., Washington D.C., 1985

Stev79 Stevens, Charles F.. "*The Neuron,*" Neuron and Synapse, 1979.

Ston87 Stone, Harold S., "*High-Performance Computer Architecture,*" Addison - Wesley Publishing Company, 1987

Sutt81a Sutton, Richard S., Barto, Andrew G. "*Landmark Learning: An Illustration of Associative Search,*" Biological Cybernetics, Vol 42 1-8, 1981

Sutt81b Sutton, Richard S., Barto, Andrew G., Brouwer, Peter S., "*Associative Search Network: A Reinforcement Learning Associative Memory,*" Biological Cybernetics, Vol 40, 201-211, 1981

Extending ASOCS to

Training-Set-Style Data

Edward F. Hart

Department of Computer Science

M. S. Degree, August 1992

ABSTRACT

This thesis studies extensions to help the basic ASOCS neural network handle training set data. Background is discussed on the need for neural networks. Instance set math for the ASOCS model is presented. Six data sets are gathered and mapped into the ASOCS neural net. Tests are done with the data sets to assist in the discovery of several problems. Some of these problems have solutions suggested in this thesis.

Mutual exclusion, arbitrary discretization, One Difference minimization, noisy and missing data, generalization using a Hamming distance metric are all discussed during the research section. As a result of these extensions to the basic ASOCS model, training-set-style data is recognized much better than without the extensions. Further research using large test sets, other generalization techniques, and different discretization techniques is suggested.

COMMITTEE APPROVAL:

Tony Martinez, Committee Chairman

Douglas M. Campbell, Committee Member

Larry C. Christensen, Graduate Coordinator