NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Incremental Inductive Learning of Discriminant Descriptions from Noisy and Incomplete Data

John Opala

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

August 1988

ISBN 0-315-49048-9

ABSTRACT

Incremental Inductive Learning of
Discriminant Descriptions from
Noisy and Incomplete Data

John Opala

A program was written by which partial discriminant descriptions of concepts can be learned incrementally by a computer from positive and negative examples. The method was applied to an Automatic Speech Recognition system to generalize over phoneme descriptions in order to produce preconditions for production rules. Data from this domain may contain errors and may not carry enough information for full discrimination of concepts. A truth maintenance system was therefore adapted to handle the non-monotonic logic. Depth-first and backtracking techniques were used to guide the search in the induction process.

for Rossana

Acknowledgements

I wish to thank all my friends, family and colleagues who have given me support during the last few years. In particular, I would like to thank Dr. Eric Regener for his advice and patience. I am also very grateful for the support, encouragement and resources offered me by the people at the Centre de Recherche Informatique de Montréal.

I am indebted to the Fonds pour la formation de Chercheurs et l'Aide à la Recherche for their financial support.

Table of Contents

List of Figures

Chapter 1.

Introduction

Learning is the process by which we acquire knowledge and expertise. It is often used as a main criterion in attributing intelligence to some entity. The role of learning as a basis for intelligent behaviour has long been recognized by Artificial Intelligence researchers. Since the 1950's computers have been used to model learning theories and algorithms. This has created the (fledgling) science of machine learning.

The efforts made in learning research can be divided into three main categories, here ordered according to their emergence historically as a center of focus [Carbonell1983]:

1. Cognitive simulation: The use of the computer as a simulation tool to investigate human learning methods. This is also known as the psychological approach. The computer's modelling capabilities are used in a sort of experimental epistemology. Some [Simon1983] consider this area of research in learning to be the most worthy of pursuit; the goal of understanding ourselves has always been one of science's most driving aspirations.

2. Theoretical analysis (the science of intelligence): Exploration of the space of possible learning methods. As with any theoretical study, efforts made here are not only interesting for their own sake, but are justified by providing a framework in which all such research can be advanced: from the practical to the speculative.

3. Task-oriented studies: The development of learning systems to improve the performance of a given task. This is also called the engineering approach. Taking the practical route, many facets of learning are discovered and their problems tackled in the service of a specific mission. Many larger systems use learning methods toward accomplishing their goal. For example, research in learning has been conducted to address the bottleneck in knowledge

acquisition for expert systems, to help develop automatic programming systems, to enable robots to adapt and refine skills, etc. In short, the aim in this approach is to transfer the burden of learning from humans (as instructors) to the computer (as student.)

Naturally, the boundaries between these objectives are not entirely well-defined.. Successful research in one of these areas often has contributed toward advancement in the others. It is in the third of these categories that the present thesis has its birth, and it is hoped that it can help add one more dot of resolution to the picture presented by the body of knowledge found in the second category.

In the practical sphere, the need for learning systems is particularly evident in systems that must acquire knowledge in forms which are only relevant to its own internal representation of its environment, forms that are alien to human thinking. In such cases, the computer needs to have the knowledge conform to its model of the world in order to perform its task. It is tedious and unnecessary for a human expert to have to compile the information in this contrived format when the computer can do the learning itself.

Relatively little effort has been expended on learning methods that are resistant to noise or that can learn from uncertain or incomplete information [Dietterich1983]. These aspects are often ignored, rendering the problem into a more tractable and ideal form, in order to create a clearer theory on learning--errors being considered a degenerate case. Yet this is precisely the sort of environment the real world presents. Undoubtedly, humans learn under such conditions, often having to unlearn some things. If a computer is to learn in this same real world, it also must be capable of dealing with these difficulties.

This thesis attempts to address these problems in its application to an Automatic Speech Recognition (ASR) system designed to be developed and operate in the real world. In this system noise comes through the data from noise (as the word is most commonly

meant) and human errors, and incomplete information is inherent in the developmental approach to the speech recognition problem that this system is applied to.

The program described here takes examples of phonemes from acoustic cues measured on sounds from the real world and attempts to generalize over them --a form of learning-- in order to characterize a class of sounds over a wider spectrum than that presented by the individual observations. These representations of the world of speech are relevant only to the ASR. For a human expert to have to do the learning described here is unwarranted, possibly less accurate and certainly less efficient. The program must provide generalizations for several classes that are as discriminant as possible, must identify plausible errors, and demarcate areas where discrimination remains unclear.

Chapter 2 gives an overview of machine learning with particular emphasis on learning from examples, as well as a formalism for dealing with non-monotonic logic. Chapter 3 describes the ASR system to which this thesis is applied and explains the nature of the data and the goals of learning in this context. Chapter 4 analyses previous work on learning for this ASR system. Chapter 5 describes the structure and workings of the learning method proposed for this thesis. Chapter 6 discusses specific problems encountered and their solutions, and chapter 7 concludes with a description of tests, evaluation of results and suggestions for further work. Appendix A provides an abridged listing of some of the results obtained for ASR, and Appendix B provides an abridged listing of results obtained from a simpler learning problem.

Chapter 2.

An overview of machine learning and non-monotonic logic.

In order to provide points of reference and familiarize the reader with the nomenclature used in this thesis, this chapter will review machine learning, giving specific emphasis on learning from examples on the theory and issues relevant to this thesis. As well, non-monotonic logic is discussed (since handling false assertions is an important aspect of this work) along with a method for dealing with it.

2.1 Machine learning definition.

As was mentioned in the introduction, learning is the method by which we acquire knowledge. Machine learning is that subset of A.I. research that is concerned with computational approaches to the problem. Simon [Simon1983] defines it thus: "Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time." This definition presents us with the idea of learning in a more tractable form in terms of computing and with it we can recognize learning systems as those whose performance improves over time. However, it does not describe what learning consists of or how it is accomplished.

The central paradigm for most learning is inductive inference. The inductive principle is the form of reasoning that allows the formation of generalities expressing a concept from individual examples of it. As was recognized by the early Greek philosophers long ago [Michalski1983], it is the method by which we come to know the primary premises of the world about us. An example best illustrates:

If after the observations: Socrates is Greek, Plato is Greek, Aristotle is Greek, and that Socrates, Plato, and Aristotle are philosophers, are made, the assertion is made that: All philosophers are Greek, then this last is said to have been arrived at by induction.

Induction as a rigorous formalism does have its problems [1]; however, leaving the deeper problems of empiricism to the philosophers, we are concerned only that induction is a practical method and its workings can be modelled on a machine.

2.2 Learning Space description.

Although the various extant learning systems differ widely in approach, expression, and application, some common aspects can be used as dimensions to describe a space of learning. This space can be used to situate, compare and analyse learning systems. Below are three such dimensions for all learning systems and a description of important points found on each [Dieterrich1983], [Carbonell1983]. Other aspects of learning that are less commonly applicable but are nevertheless relevant to this work will also be explored.

2.2.1 Methods of acquiring knowledge.

Points along this dimension refer to general methods of acquiring knowledge. They are here ordered in increasing independence (one might say intelligence) of the learning system. This is to say an increasing amount of inference is used, coupled with a decreasing dependence on a teacher.

2.2.1.1 Rote learning.

This is akin to memorization for humans or direct programming for computers. All information is directly assimilated without translation to another form, nor is any inference or analysis performed. A computer that is simply acquiring information (e.g. a program) can be

said to be learning; however, this is a degenerate form. The only effort expended on its part is that of storage; nevertheless, new knowledge is assimilated.

2.2.1.2 Learning from instruction.

Here the knowledge is presented directly as well, but only from a higher level of abstraction, so that the learner must translate the information into its own internal representation. This is the form of learning most people associate with the education process.

2.2.1.3 Learning by analogy.

In this form of learning, new knowledge is gained by adapting existing knowledge (of skills, concepts etc.) that is similar to that of the present goal. See [Carbonell1983a] for example.

2.2.1.4 Learning from examples.

This area of learning is the most extensively explored. Examples and counterexamples of a concept are provided to the learner so that it may form a generalization. This is the method adopted in this thesis. It is reviewed in greater detail below.

2.2.1.5 Learning by doing.

Learning in this case involves some procedure that is successively improved based on experience. Applications include learning of heuristics, game playing, robot control.

2.2.1.6 Learning from observation and discovery.

In this method, inductive inference is made without a teacher. The learner assimilates knowledge (classifying to form taxonomies, forming theories, etc.) by simply observing its

environment or by actively experimenting with it.

2.2.2 Knowledge Representation.

Naturally closely linked to the goal of the learning system, this dimension classifies learning systems according to the form in which knowledge is expressed. Representations commonly used include:

Parameters in algebraic expressions

Decision trees

Formal Grammars

Production Rules (of particular relevance to this thesis)

Formal logic-based expressions

Graphs and networks

Frames and Schemes

Computer programs and other procedural encodings

Taxonomies

Multiple representations.

[Carbonell1983].

2.2.3 Domain of application.

Learning systems can be classified by the domain to which they are applied. Systems can range from being generally applicable, to being inextricably linked to tasks in specific applications such as, game playing, medical diagnosis, natural language processing, speech recognition, and so on. The list of domains to which learning methods have been and are being applied is long and growing.

2.3 Learning as search.

As with most A.I. endeavors, the role search plays in learning is an important one. In fact, the entire process of inductive learning can be viewed as a state space search [Michalski1983]. Observational statements (the initial state), expressed in some formalism, are combined or transformed by the application of operators to create inductive assertions, which can then be modified similarly to create new assertions. Operators and statements to which to apply them are chosen in a directed manner according to some control strategy. Processing continues until a goal state is reached (a set of general inductive assertions), which implies the observations, conforms to the criteria specific to the particular learning task, and maximizes some preference criterion.

This association with the more familiar theory of search provides a background on which processing methods of learning systems can be characterized and compared.

2.4 Learning from experience.

This general label of inductive learning covers the last three methods described above. Learning from examples can be viewed as a subset of this larger task. The entire task of knowledge acquisition is here divided into three main operations:

1. Clustering

2. Characterization and

3. Storage/indexing.

The object of clustering is to identify observations as belonging to a certain class. Given only a description, the learner must come to recognize the observation as an instance of some specific concept. Naturally, any such classification represents a hypothesis. Some advances in this area are found in the Conceptual Clustering method by Michalski and Stepp [Michalski1983a] for example.

Characterization is that part of the problem involving the formulation of general descriptions of the concepts to be learned. Having clustered the observations into classes, the learner must generalize them to create definitions for each class.

Storage and indexing involve the organization of the new knowledge for efficient retention and retrieval. This is particularly important in systems where the learning material covers a big space.

2.4.1 Learning from examples.

This type of learning is a degenerate form of learning from experience. In this case, the clustering portion of the learning task is accomplished by the teacher, who preclassifies the observations. Typically, systems performing this type of learning cover only a relatively small set of concepts and the storage/indexing task is trivial. The remaining task then, that of characterization, is the central endeavor of learning from examples.

The problem is most simply stated as follows: Given a set of positive and possibly negative examples of some concepts, the goal is to generate descriptions for each concept that cover all of the positive examples and none of the negative examples.

A positive example is one that describes an instance of a given concept, and is denoted thus:

$$a ::> A$$

where observation 'a' is an instance of class 'A'. A negative example is one that describes an instance of something other than a given concept. i.e.:

$$b ::> \sim A$$

This notation also applies to generalizations, which are not 'instances' of a concept, but whose descriptions can also be said to fall within a concept.

The first constraint in the description of the goal above, that of complete coverage of positive examples for a concept, is called the completeness condition. The second--no coverage of negative examples-- is called the consistency condition.

In the remainder of this work, the terms 'example' and 'observation' are considered synonymous, likewise the terms 'concept' and 'class'.

During the course of learning, operations are applied to observations and other assertions created during learning. These operations create new assertions which can be considered more or less general than others. This is to say that learning is achieved in a space where a partial ordering based on relative generality exists. Such a relation is denoted thus [Michalski1983]:

$$a \mid< b,$$

meaning that assertion 'a' can be generalized to, or is less general than assertion 'b'. It follows that 'b' is more general than 'a'. Conversely,

$$b \mid> a,$$

meaning that assertion 'b' can be specialized to, or is more general than assertion 'a'.

The issues central to learning from examples are those of representation of knowledge, type of concept descriptions sought, methods of generalization, and search control. These and other issues are discussed below.

2.4.2 Representation of knowledge.

2.4.2.1 Representation language

Of critical importance to the learning task is the choice of representation language in which observations and generalizations are expressed. Naturally the choice is dictated by the nature of the learning task. It is important that the language is rich enough to be capable of

easily expressing all the assertions that will be necessary, and be able to do so in sufficient detail to apprehend the crucial features. For example, while attribute descriptions (descriptions specifying global properties of an object) can be sufficiently captured by propositional logic or a simple extension thereof, structural descriptions (descriptions of composite structures consisting of various components) must be capable of describing relations between components and therefore need the more expressive power of predicate logic or some similar form. However, the language chosen should not be so rich as to include irrelevant detail. The scope of allowed forms and the modes of inference must be clearly delineated.

As was shown above in Section 2.2.2, the forms are varied and provide a key for classifying learning systems. One of the most often used forms is the predicate calculus, or some subset or extension thereof adapted specifically for use in inductive systems. It is chosen for its expressiveness and clear syntax and semantics. Such a form, 'annotated predicate calculus' (APC) [Michalski1983], is used below to describe some common generalization rules. Typically, assertions in learning are described by conjunctions and disjunctions of descriptors. For example, the description of an apple might be expressed as:

$$\begin{aligned}&(\text{shape(object)} = \text{spherical} \wedge \text{color(object)} = \text{green} \wedge \text{taste(object)} = \text{sour}) \vee \\ &(\text{shape(object)} = \text{spherical} \wedge \text{color(object)} = \text{red} \wedge \text{taste(object)} = \text{sweet}) \vee \\ &(\text{shape(object)} = \text{spherical} \wedge \text{color(object)} = \text{brown} \wedge \text{texture(object)} = \text{wrinkled})\end{aligned}$$

2.4.2.2 Descriptors

The descriptors are those observable features and measures of the examples used for learning. These are selected by the teacher. The choice of descriptors (itself an integral part of the learning task) should naturally be based upon their relevance to the subject of learning.

Learning is conducted in the space defined by these descriptors. The descriptors themselves have attributes (type, domain, interrelationships with others, etc.), the nature of which can be exploited to constrain the description space and define what movements may be made in it. In other words, the learning system can be tailored according to the specific nature of

the learning task through its expression in the descriptors and their peculiarities. This is what Michalski [Michalski1983] calls a part of the problem background information–a set of assumptions and constraints imposed on the observational statements and generated candidate inductive assertions, and any relevant problem domain knowledge. As well as having influence in the expression of descriptors, such problem background information affects the forms of generalization and search control. Problem background knowledge is one of the components of the definition of Michalski's general paradigm for inductive inference.

With respect to the relevance of descriptors, three types of learning can be delineated:

1. Learning with completely relevant descriptors; The descriptors capture the essential properties of the objects of learning. Only the information necessary to characterize the desired concepts is available.

2. Learning with partially relevant descriptors; Observations include redundant or irrelevant information along with relevant information. The irrelevant information muddles the inductive process. For example, when learning how to distinguish between cars and bicycles, information in the examples about weight and number of wheels is relevant. Information about color and age of vehicles can only confuse the learner. The learner's task in this case is to determine the relevant information and to carry on induction with that subset of the input.

3. Learning with indirectly relevant descriptors; In this case the observations contain no descriptors immediately relevant to the learning task. However, relevant information can be derived from some of the initial data. As a simple example, if speed is a relevant descriptor to some concept but is unavailable, and distance and time are not directly relevant, but are nevertheless provided, the speed might be computed.

This distinction of learning based on relevance of descriptors defines the premises for two forms of induction by which learning systems are characterized. In the first, selective

induction, the information used to characterize the concept is found in the observations. Such systems must deal with the problem of selecting the relevant information. See [Quinlan1983] for instance.

In the second, constructive induction, information in the observation is transformed, creating new descriptors and thereby altering the representation of the learning space. [Lenat1983] is an example.

2.4.2.3 Domain and type of descriptors.

The domain is the set of all possible values a descriptor can adopt. With information about the domain of descriptors, a learning system can be constrained to operate only within this bounded space.

The structure of the information carried by a descriptor, as is given by its type, is useful information to a learning system because it determines what operators can be applied to the descriptor. Here we cover three basic types:

1. Nominal descriptors.

The values of these are independent symbols or names. There is no structure organizing them. For example: Predicates and any n-ary function whose range is an unordered set such as Name(person) or Haircolor(person), etc.

2. Linear descriptors.

Values of these are organized in a totally ordered fashion. For example, age, temperature, and weight are linear descriptors.

3. Structured descriptors.

Values of such descriptors are organized as tree oriented graphs that capture the pro-

perty of relative generality between values. That is, parent nodes are more general than their children nodes. For example, for the descriptor "place", we could have "Canada" as a parent node and the names of the ten provinces and two territories as its children nodes.

2.4.2.4 Other constraints

Many other constraints on descriptors can be imposed, depending on the problem task. Here is a brief description of three common constraints.

1. Interdependence among values.

This can occur when one descriptor specifies a state and another characterizes that state. For some values of the first descriptor, the second may be redundant, irrelevant, or contradictory. For example, if we are trying to distinguish many types of faces, then when the descriptor Hair-length(face) takes on the value 0 or 'beardless', then the descriptor Hair-color is irrelevant.

2. Properties of descriptors.

Particularly with descriptors that describe relations among objects, there may exist certain general properties such as symmetry, transitivity, etc. The reflexivity of the relation sibling-of, or the transitivity of the relation heavier-than are examples of such general properties.

3. Interrelationships among descriptors.

Specific to the problem, some special relationships may exist between descriptors to constrain their values. For example, if the learning task involves insects, the following might be expressed;

$$\text{for every P, number-of-legs(P)} \geq \text{number-of-body-segments(P)}$$

2.4.2.5 Descriptions Sought.

The goal of the learning system is to produce some form of characterization of the concepts presented to it. The form of these descriptions, usually dictated by the use to which they will be put, provides another classification by which learning can be described.

1. A Characteristic description is intended to describe a class of objects so that they can be distinguished from all other possible classes. For example, the characteristic description of apples would discriminate any apple from all things that are not apples. Such a description is typically expressed as a conjunction of basic properties common to all (known) objects in the class. A Maximal Conjunctive Generalization (MCG) [Hayes-Roth1983]is the longest such expression. That is, it gives the most descriptive account of the class, identifying all the properties that hold for every member of the class.

2. A Discriminant description describes a class in the context of a fixed set of classes. The characterizations need only state the properties that are sufficient to distinguish members of one class from the rest. The Minimal Discriminant Description is the shortest such description. It provides the minimum information necessary for an object to be recognized as a member of one class and not any other. Such descriptions are commonly represented as conjunctive expressions or disjunctions of conjunctive expressions.

3. Other more complex types of descriptions are learned through more powerful inductive methods that accomplish descriptive generalization [Lenat1983]. For instance, a Taxonomic description is a structured description of a class that is partitioned into subclasses. Naturally, the examples in learning tasks that attempt to determine a taxonomy do not necessarily belong to a single class. Taxonomies can be flat or hierarchical, and their descriptions are fundamentally disjunctive.

An important distinction between learning systems addresses the issue of whether there is only one concept for which a description is sought, or whether the learning system is

simultaneously acquiring several concepts.

In single concept acquisition, one can distinguish two forms based on the data available. In the first, only positive examples are presented to the learner. Naturally, in this case the consistency constraint does not apply. The major problem however, is that evidence from examples cannot provide any indication of overgeneralization. Constraints on generalizations must be provided to the program as problem domain knowledge or in terms of the forms of the generalizations (e.g., where MCG is the goal.) In the second form we distinguish here, negative examples, also known as counterexamples, are also provided to the learner. Such situations provide means of curbing overgeneralizations by enforcing consistency. Should a generalization be made that is so wide as to encompass a negative example, it is an overgeneralization. Of particular interest is the variety of counterexample known as a near miss [Winston1975]. This is an example that just barely fails because of some small variation in the description. Such counterexamples provide very clear constraints during the generalization process by demonstrating very specific distinctions.

Systems learning multiple concepts naturally have both positive and negative examples available, as a positive example in one class may represent a negative example in another. This is only the case, however, if the classes are mutually disjoint. Some learning problems have concepts that overlap, where an observation can be a member of one or more classes. For example, if the subject of learning involves the distinction of mechanical difficulties in cars, and the observational examples consist of lists of symptoms such as black smoke, no ignition etc., then it is certainly possible for a car to have several things wrong with it, and in such a case one would expect the symptoms to be consistent with all the problems involved. In cases of overlapping classes, the consistency constraint must be relaxed.

2.4.3 Generalization Rules.

The following is a short description of some of the common generalization rules found in inductive learning systems. In the characterization of learning as a state space search, such inference rules are considered to be the operators that transform the current state into the next. These inference rules can be divided into three types:

1. *Generalization rules* that transform descriptions into more general descriptions. These new descriptions imply the descriptions from which they were derived. e.g.

   if e ::> K and f ::≥ K get transformed into g ::> K by a generalization rule,

   then g → e and g → f.

2. *Specialization rules* that perform the opposite transformation, generating logical consequences (more specific descriptions) from general descriptions.

3. *Reformation rules* that transform descriptions into other logically equivalent ones.

Specialization and reformation rules consist of the familiar deductive inferences. They are truth preserving; if some description is true, then its specialization or transformation is true as well. However, generalization rules, as a consequence of induction, are falsity preserving; If a description is false, then its generalization is false also. The reverse--if the generalization is false, then the original description is false--does not hold. For example, in deduction the following is truth preserving: All men are mortal; Socrates is a man; therefore Socrates is mortal. In contrast, the induction "Socrates is Greek; Plato is Greek; Descartes is Greek; so all philosophers are Greek" is falsity preserving; because the third observation is false, so necessarily is the generalization based on it. To illustrate that the reverse does not hold, if this generalization were based only on the first two observations, and was found to be false, this would not discredit those observations. To say that "all philosophers are Greek" is false does not imply that "Socrates is Greek" or "Plato is Greek" is false.

Note that because of the induction, these generalizations may not be true, however they do express an aspect of plausibility. Assertions created by generalization rules can be compared to data or tested empirically for validation. The search for the best correct assertions created with these rules constitutes the core of the learning process.

Reformation rules are used in constructive generalization methods, and will not be covered here. As well, specialization rules will not be covered explicitly, as they can be viewed as the reverse of generalization rules and can be found in standard texts on logic. Only selective generalization rules used for concept acquisition are described below. We further restrict the scope by examining only generalization rules that transform one or more statements into a single more general statement.

In concept acquisition, the term 'more general' has a simple set-theoretic interpretation: a description is more general if it is satisfied by a larger number of objects.

The language used below to express the rules is taken from the Annotated Predicate Calculus (APC) described by Michalski [Michalski1983]. The language conforms to all the notation used so far in this thesis and its interpretation ought to be self evident.

2.4.3.1 Rules.

1. Dropping Condition rule.

$$Ctx \wedge S ::> K \mid< Ctx ::> K$$

where K is a class, Ctx is some description (here meant to denote context), and S is an arbitrary predicate or logical expression.

This rule (the most commonly found) states that a description can be generalized by removing a conjunctively linked expression. For example, animal ::> person is a generalization of animal and two-legs ::> person.

2. Adding Alternative rule.

$$Ctx1 ::> K \mid< Ctx1 \vee Ctx2 ::> K$$

A description can be generalized by adding an alternative (using disjunction). For example, apples can be round and red or green in color which could be expressed (allowing internal disjunction) as

(shape = round) and (color = red) ::> apple

|<

(shape = round) and (color = red or green) ::> apple

Important special cases of this rule involve the extension of the scope of values for one descriptor as additions of alternatives. These take on different forms depending on the type of the descriptor whose scope is extended.

2a. Extending Reference rule.

$$Ctx \wedge [L=R1] ::> K \mid< Ctx \wedge [L=R2] ::> K$$

where $R1 \subset R2 \subset Dom(L)$. Dom(L) is the domain of L. L is a term that acts as descriptor when it adopts values, and the sets R1 and R2 contain values L can adopt (to satisfy membership in K). R1 and R2 are called references. A description can be generalized by enlarging the reference of a descriptor.

In particular, if in this case R2 = Dom(L), then the descriptor [L=Dom(L)] is always true and can therefore be deleted. Under these circumstances, this rule is the same as the dropping condition rule.

2b. Closing Interval rule.

$$\left.\begin{array}{l} Ctx \wedge [L=a] ::> K \\ Ctx \wedge [L=b] ::> K \end{array}\right| < Ctx \wedge [L=a..b] ::> K$$

where L is an ordered linear descriptor and a and b are values in its domain. If two descrip-

tions differ only in the value of one linear descriptor, then they can be generalized to a description where the reference of the descriptor in question is an interval linking the values.

e.g. wheels=4 ∧ doors=2 ::> car
wheels=4 ∧ doors=4 ::> car
|< wheels=4 ∧ doors=[2..4] ::> car

2c. Climbing Generalization Tree rule.

Ctx ∧ [L=a] ::> K
Ctx ∧ [L=b] ::> K
. . .
. . .
Ctx ∧ [L=i] ::> K
|< Ctx ∧ [L=s] ::> K

where L is a structured descriptor and s is the nearest ancestor to nodes a, b, ..., i in the generalization tree L represents. Naturally, the rule only applies to descriptions with such descriptors.

e.g. season=winter ∧ site=Montreal ::> cold-place
season=winter ∧ site=Trois-Rivieres ::> cold-place
season=winter ∧ site=Quebec-city ::> cold-place
|< season=winter ∧ site=Quebec-province ::> cold-place

3. Turning Conjunction into Disjunction rule.

F1 ∧ F2 ::> K |< F1 ∨ F2 ::> K

where F1 and F2 are arbitrary descriptions. A description can be generalized by changing a conjunction to a disjunction.

4. Inductive resolution rule.

P ∧ F1 ::> K
~P ∧ F2 ::> K
|< F1 ∨ F2 ::> K

where P is some predicate and F1 and F2 are arbitrary descriptions. This rule is taken from the resolution principle of deductive logic by interpreting formulas as concept descriptions.

For example,

$$\left.\begin{array}{l}\text{cold} \wedge \text{refreshing} ::> \text{swimming-pool} \\ \text{hot} \wedge \text{relaxing} ::> \text{swimming-pool}\end{array}\right| < \text{refreshing} \vee \text{relaxing} ::> \text{swimming-pool}$$

5. *Extension Against rule.*

$$\left.\begin{array}{l}\text{Ctx1} \wedge [\text{L}=\text{R1}] ::> \text{K} \\ \text{Ctx2} \wedge [\text{L}=\text{R2}] ::> \sim\text{K}\end{array}\right| < [\text{L}\neq\text{R2}] ::> \text{K}$$

where R1 and R2 are disjoint sets. Given a positive example of a concept with a descriptor taking values from one set, and a negative example with the same descriptor taking values from a second disjoint set, the generalization describing objects with this same descriptor not having any values from the second set can be made (generalization of the class of the positive example.) For instance,

$$\left.\begin{array}{l}\text{poor} \wedge \text{health=good} ::> \text{happy} \\ \text{rich} \wedge \text{health=bad} ::> \sim\text{happy}\end{array}\right| < \text{health}\neq\text{bad} ::> \text{happy}$$

This rule is useful in systems that try to determine discriminant descriptions (see Section 2.4.2.5 above) for it provides a single criterion by which an object can be identified as being a member of a certain class or not.

2.4.4 Search through a description space

Despite the constraints on the learning description space incurred from representation of the learning problem, the space of possible descriptions in a concept acquisition task usually remains extremely large, and the attainment of a goal description through the successive application of operators on states in this space must be conducted in an organized manner. The directed exploration of this space then is a form of search, and some of the methods and issues involved with searching as applied to inductive learning are reviewed below. Note that many of the different aspects investigated below are related or closely linked. For example,

there is a strong association between data-driven methods and incremental learning.

2.4.4.1 Direction of search

As we have mentioned, descriptors of concepts are partially ordered according to relative generality. This ordering implies three basic approaches to goal attainment, identified by the directions taken to achieve it.

1. General to Specific.

Methods working in this direction start with (overly) general descriptions and employ specialization rules to attain more specific hypotheses about the goal.

2. Specific to General.

In this direction, methods begin with specific descriptions and hypothesize about the goal by generalizing.

3. Bi-directional.

Methods using a bi-directional approach attempt to converge on a goal description from both the specific and general sides, thereby defining bounds on the concept.

2.4.4.2 Data-driven and model-driven methods

Inductive learning methods can be divided into data-driven and model-driven strategies for conducting search. Some systems employ a mixture of both.

Data-driven methods, also known as bottom-up methods, apply the generalization rules (or specialization or transformation rules) as operators to the input observational statements in order to generate new hypotheses. They use data to gradually converge on a goal. Examples include Winston's Blocksworld [Winston 75], Hayes-Roth's Sprouter, and Vere's Thoth

(see [Dietterich1983]).

Model-driven methods, also known as top-down methods, supply other information to the operators. The observational statements are used only to test the hypotheses generated. Such a strategy is also called a Generate and Test method. An example of such a system is Induce [Dietterich1983].

A mixed strategy might employ a data driven approach to positive examples and a model driven approach to negative examples.

2.4.4.3 Incremental and non-incremental methods

The way that observational data is presented to a learning system characterizes the conduct of the search. This can be done non-incrementally (one-shot) or incrementally.

Non-incremental methods have all the input examples available at the outset. This allows them to use statistical methods to evaluate competing hypotheses. Such a procedure can perform analysis to identify noise and can organize the data so as to maximize the speed of convergence during generalization. That is, there is less exploration of bad paths resulting from poorly chosen premises.

Incremental methods process examples one at a time. Systems employing this method must formulate generalizations consistent with data encountered so far, and must subsequently refine these generalizations after considering additional examples. By their design, these systems respond well to new information, but the speed of convergence on the concept(s) is often dependent on the ordering of the examples. Exceptional cases in the data, and in particular noisy data, can result in the formation of bad premises that make recovery difficult. In fact, such systems can produce different solutions with different orderings of the data.

These systems most closely resemble human learning, and allow for the use of partially

learned concepts. For example, if interaction with the teacher is possible, such systems can present partially learned concepts to show the strengths and weaknesses in the concept formation, and allow the teacher to guide the choice of new examples and focus attention on poorly resolved aspects of the concept.

2.4.4.4 Search control strategies

The search of the description space where competing hypotheses are evaluated and chosen for further exploration is a matter of program control taken from conventional search procedures. The methods are as varied as there are searching paradigms. A few basic approaches are described here for their relevance to this thesis.

A breadth first search will generate a whole set of competing hypotheses at each level of complexity. Such systems have large memory requirements but do not need to backtrack from unfruitful explorations. Because all paths to a solution are explored to some degree, alternative solutions--in the case of learning this reads as alternative descriptions--are more thoroughly examined and therefore more easily attainable. Thus such systems can often learn disjunctive concepts.

Depth first methods consider only one hypothesis at a time, and therefore have less taxing memory requirements than breadth first searches. However, a depth first approach to search might require much backtracking to recover from dead-end paths. Although rapid convergence on a concept description satisfying the goal is possible, such systems can usually only learn conjunctive concept descriptions.

Both these search methods and their variations can be exhaustive or they may be guided to the goal by heuristic evaluation functions, thereby taking advantage of some of the problem background knowledge. For example Michalski [Michalski1983] in defining his Star Methodology of inductive learning describes a Lexicographic Evaluation Function (LEF) as a set of

criterion tolerance pairs that is used to prune hypotheses (inductive assertions) during search. A score is computed for each criterion, and starting with the first criterion, only the best scoring assertions, or those within the given tolerance from the best, are retained. Pruning continues to the next criterion until the number of hypotheses is manageable from the searching point of view. Such a function is a formalization of the characteristics desired for a goal description. Naturally, the criteria chosen depend on the purpose of the learned descriptions. The criteria can include the degree of fit between the assertion inductively arrived at and the observations made, the readability for human interaction considerations, and the costs in computation and storage.

2.4.4.5 Intra-operator Search

What has been covered so far concerning search is applied at the level of choices between whole assertions to generalize with. Search can also be involved at a lower level in the process of comparing, contrasting, or combining assertions during generalization. The variety of descriptors and the methods of combining them also defines a large search space: i.e. given two assertions, it might be possible to generalize them in many ways. Search at this level is called Intra-Operator Search.

2.4.4.6 Simplifying Assumptions

Most systems that learn from examples employ some simplifying assumptions about the task in order to reduce the problem to a more tractable form. Unfortunately, some of these assumptions make the systems unusable for tackling real-world problems where the assumptions can not be realistically applied. Some of these assumptions are:

That the concept can be described with a set of necessary and sufficient conditions. That is to say that the concepts are clear cut and can be expressed.

That the concepts do not overlap.

That the concepts do not change over time.

That no noise is present in the observations. No "positive" examples are actually negatives, and vice versa.

That the language and the descriptors chosen for representation are sufficient for description of the concept.

That a concept can be described with only conjunctive conditions.

Using such assumptions, systems can learn inductively with the rather simple completeness and consistency conditions. If however, some assumptions must be dropped, these conditions must be relaxed. For instance, with respect to the completeness condition, the goal of learning is to create a hypothesis H that tautologically implies the facts F. That is, F is a logical consequence of H, that is, $H \rightarrow F$ is true under all interpretations. If however, noise can possibly be present in the facts, the system must search for a hypothesis that weakly implies the facts. That is, F is only a plausible or partial consequence of H. For example, partial hypotheses could account for some but not all the facts.

2.5 Non-monotonic logic.

Because the system proposed here must operate in a real world environment, it is quite possible that errors might be present in the data. Since the strategy chosen for this program is incremental, the errors cannot be determined, or even guessed at, a priori without some knowledge about the problem. Therefore, every datum is at least initially treated as fact, or rather, is accorded some belief: as premises, all observations are assumed to be true. Upon further assimilation of data, evidence may show some beliefs to be poorly founded and must therefore be disbelieved according to some method of measuring the plausibility of an item in the data based on the rest of the body of data.

The system of evidence can also show that some data previously disbelieved is currently quite likely to be valid. These then must be re-believed. In fact, it is possible for assertions to flip-flop between belief and disbelief many times, based on competing evidence for and against them that grows over time.

The consequences of having basic premises change truth values during the operation can be dire for any system, deductive or inductive. For if the premises can change, all the conclusions based upon them must also be reevaluated. In such situations, it must be possible to propagate the consequences correctly and efficiently, and perhaps to minimize any duplication of effort in further search toward a goal.

This is to say that the logic employed is non-monotonic. Most reasoning done by computers is monotonic, in the sense that the number of true statements is strictly increasing over time. New statements are added and conclusions are derived from them, but this does not cause any previously known statement to become invalid. In non-monotonic reasoning, this is not the case. What was true, or assumed true through the use of some default reasoning, can become or be shown to be false, and all the reasoning based upon the belief in its validity must be undone.

There are several methods of dealing with non-monotonicity including fuzzy logic, probabilistic methods, and backtracking search. Because of the incremental approach adopted for this system adopted in this thesis, a backtracking search method was chosen as best suited to deal with the non-monotonic nature of the task. Fuzzy logic and probabilistic methods are more often associated with non-incremental approaches. One specific search mechanism, the Truth Maintenance System (TMS) [Doyle1979], described briefly below, was adapted to fit the needs of the task defined in this thesis. A description of the task involved in this thesis is found in chapter 3. Chapter 5 provides a description of the system and the solution of the task.

2.6.1 Truth Maintenance System

Most often the disbelief of some given premise occurs indirectly because the addition of new knowledge created a logical conflict in some derived assertion. Upon discovery of a conflict, the system backtracks to the point where the assumption was made that the given premise is valid, and is able to continue its operation in a logically consistent manner by reversing that assumption.

For example (fig. 1), if a program were trying to determine a time and place for a conference for a group of professors based on convenience and economy, it might attempt to establish the time first, then the place, and then test its solution. Suppose it chose Christmas as the time for the conference and then decided that the location would be Montreal based on the distances each professor would have to travel (and independent of the choice of time). It might discover or be told that this solution fails because no one wants to go at Christmas. It must now retrace its steps to the point where it decided on the time and start recomputing the time and location.

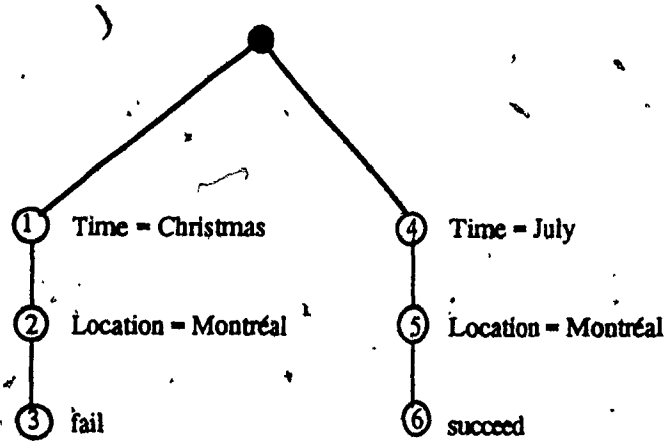

Figure 1: Search and Backtracking.

Thus, one method of dealing with non-monotonicity is straightforward tree search with backtracking. However, as seen in the example, the program after backtracking has lost all

the reasoning that was made after the assumption that the time be Christmas, even though this reasoning did not depend on that assumption. It must now go over this same ground in recomputing a solution.

By withdrawing statements based on the order of their presentation, rather than responsibility for inconsistency, much effort is wasted. One method of dealing with this problem is called Dependency Directed Backtracking, where assumptions and inferences can be inserted into and withdrawn from a data base directly.

Since the withdrawal or disbelief of statements can have far-reaching consequences in a knowledge base (i.e., all that was based on withdrawn statements must be altered), it is useful to record with each assertion generated its logical ancestry, that is, the list of other statements on which it depends for validity. This record provides a method of efficiently propagating changes in a non-monotonic system and of implementing dependency directed backtracking.

The TMS provides a method of recording the dependency of assertions and employs dependency directed backtracking to resolve conflicts. In TMS, all assertions are recorded as nodes that are accorded a status value; 'in' or 'out' indicating current belief or disbelief respectively in their validity. As well, accompanying each node is a support list that records its logical dependency on the validity accorded to other nodes. That is, the support list describes the derivation of an assumption indicating which of the other assertions must be believed and which disbelieved for the present to be valid. Thus the support list is actually two lists; An 'IN list' naming the other nodes that must be 'in' for the current assertion to hold, and also an 'OUT list' naming those that must be 'out' for the current assertion to hold. Premises as well as derived expressions are represented in this way. For example, we may have the following nodes:

(1) Time is July (SL () ())

(2) Weather is sunny (SL () ())

(3) Temperature is hot (SL (1,2) (4))

(4) Place is Antarctica (SL () ())

where status, in or out, of the assertion that it is hot (node 3) depends on nodes 1 and 2 being in and node 4 being out. As nodes 1, 2 and 4 are premises, they currently have no support, however this may change as more evidence is provided.

The list of nodes necessary for support need not be exhaustive. If only the assertions from which a given inference is directly derived are included on the support list, the inference is indirectly linked to all the rest of the assertions on which it depends.

When an inconsistency is discovered, dependency directed backtracking is invoked, the aim of which is to trace back to premises and find one (or a few) premise(s) which when changed in status, make(s) the conflict disappear. The results of these transitions are propagated throughout the knowledge base so that it can be in a consistent state. For example, if it were in fact the case that nodes 1, 2 and 3 were believed and we had further:

(4) Place is Antarctica (SL () (5))

(5) Place is Florida (SL () (4))

with node 4 'out' and node 5 'in', and then the program discovers that it is cold causing a conflict with node 3, it might then disbelieve that it is July, forcing the disbelief that it is hot. As well, all other assumptions based on its being July having node 1 in their 'in' lists, would be forced 'out' as well. Those based on node 5 and/or node 2 and unrelated to node 1 would remain intact; the reasoning involved with them need not be undone and recomputed, only that on which the contradiction depended would have to be altered.

Inferences that have gone 'out' are retained in case some new information should conflict with whatever information forced them to go out in the first place. In this case, should the assertion that forced others 'out' itself go 'out', the others can be rebelieved directly

without rederivation.

TMS is a tool to be used as a consistency watchdog for a parent program operating in a non-monotonic environment. The creations of assertions based on the premises are the responsibility of the parent program. The validity accorded to the premises is also the responsibility of the parent program; however, TMS handles all the necessary changes in status of the assumptions and the consequences thereof when inconsistencies arise.

TMS is therefore useful to a learning system where the data is not certain. Although the logical method is inductive rather than deductive in a learning system, the basic operation of TMS can be adapted where observational statements are taken as premises, generalizations are used as derived assertions, and overgeneralizations or bad generalizations represent the conflicts.

TMS has been used by Whitehill [Whitehill1980] in learning, and the general methods used by him have been adapted by Gilloux [DeMori1987] and have further evolved into the methods proposed here. The simple description above of TMS reflects the main paradigms retained and adapted to this program.

Chapter 3.

Learning for Speech Recognition

3.1 Automatic Speech Recognition.

The aim of automatic speech recognition is to be able to recognize the speech of many speakers and many speaking styles and to do so quickly and effectively. Recognition of a large vocabulary in a multi-speaker environment is a very complex task. The approach taken by the Automatic Speech Recognition system (ASR) by DeMori et al. [DeMori1987] to which this learning program is directed, is to create a knowledge based procedural network.

In a procedural network applied to speech recognition, procedures are associated with transitions between states of a network. A procedure computes the similarity of an input signal to the elements of a set of prototypes. In a knowledge based approach, speech signals are described as sets of acoustic properties. A knowledge based approach is so named because the procedures are infused with knowledge about recognition (phonetic knowledge) in order to achieve better results. Phonetic knowledge can include algorithms to extract acoustic cues from input waveforms and contextual constraints, for example. In this ASR, the procedures are perceptual plans that must be capable of implementing various recognition strategies that direct the focus of attention based on speaking styles. Speaking styles represent groups of differing pronunciation methods. Speakers of different mother tongues often speak in different 'styles'. The organization of these plans into a procedural network, as an Augmented Transition Network for instance, and the nature of its operation, e.g. data-driven or model-driven, are issues of ASR at a level well beyond the scope of this thesis.

The perceptual plans themselves, for fast execution during recognition, must be well organized. In this ASR the organization is a Network of Action Hierarchies that operate according to an elaboration-decision paradigm. An elaboration phase consists of the execution of procedures that extract information about the signal under examination. In a decision

phase, signal descriptions constructed from the elaboration are used to control the choice of actions. These actions either generate phonemic hypotheses, or instigate further elaboration-decision cycles.

Thus during recognition, the system proceeds as follows: An input signal is received and analysis procedures are performed on it to extract acoustic cues. Acoustic cues are measures of features and properties of the input signal. Next, a decision phase is begun where competing preconditions of rules are compared to the data descriptions obtained so far. A fuzzy algebra is used to order the candidates of partially matched preconditions. If two or more preconditions are satisfied with the same degree of confidence, they define an Active Confusion Set. The most similar precondition to the current data description acts as a rule to determine the next action to perform. This action is supposed to discriminate best the members of the active confusion set, or if there is no confusion, it will produce a hypothesis proposing a classification of the sound. Thus either a hypothesis of the signal is produced, or another elaboration-decision cycle begins, where elaboration consists of more signal processing and property extraction to sharpen the focus (see fig. 2). This process is repeated as often as needed to generate required hypotheses or until all the choices encountered in the cycles are exhausted. This last eventuality indicates that the recognition system needs to undergo more development; new procedures might be written for a better description of the signals during some elaborations, or more learning might be necessary to create a better classification (from which the rules mentioned above are generated).

The generation of the network described for the ASR is intended to be accomplished in a semiautomatic manner. Although an expert is ultimately responsible for conceiving perceptual plans, he does so with the aid of some high level tools. In particular, a Plan Generating Expert System is used in the development of this ASR in order to help generate the procedures used in the elaboration phases. An inductive learning program is used to generate

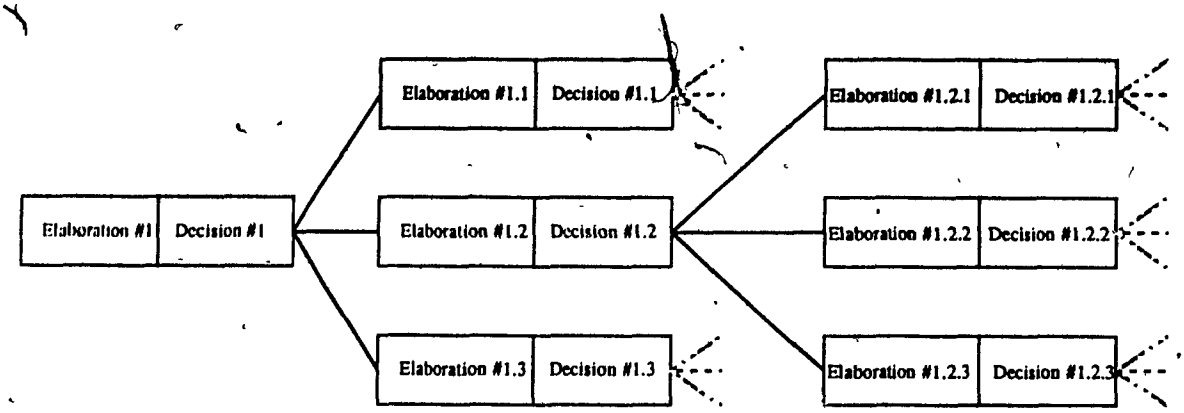

Figure 2: Elaboration-Decision cycles.

the rules mentioned in the decision phases. The design and creation of a system aimed at tackling such a large and complex problem can not be achieved directly. Rather, it must evolve through successive refinements. In fact the entire design is viewed as a planning activity whose goal is that of producing discriminant descriptions for the phonemes to be recognized. The classification of descriptions must be learned from experiments (i.e. learned inductively), and the discrimination power of a classification can not be predicted; it must be evaluated experimentally. Refinement of a plan occurs when unacceptable recognition confusions occur. Refinement is the expansion of an action into a more detailed plan.

Throughout this development stage, a human (an expert) interacts with the planning and learning systems, guiding their operations according to his or her knowledge of phonetics. In the learning system intended for development of the ASR, interaction with the teacher, apart from his choice of descriptors and supply of data, takes the form of preliminary alignment of the data and the interpretation of the results of learning for further realization of the system.

The planning and creation of the elaborations, involving as it does many complex methods, is beyond the scope of this thesis and unnecessary for its explanation, and so the elaboration facet of the ASR will not be covered in any greater detail.

3.2 The Learning Task.

As was hinted above in the discussion on decision phases, the learning system must provide a classification of phonemic hypotheses or active confusion sets. That is, it must take data in the form of sets of acoustic cues (along with the name of the sound they represent, the class) and generalize them. The generalizations are descriptions of a class that represents either a specific hypothesis, or a set of classes (active confusion set which might be discriminated through further elaboration). These generalizations then can be interpreted as rules, where the descriptions act as preconditions, and the classes each describes act as the actions (hypotheses or sets requiring futher elaboration). During recognition, cues taken from the input signal are compared with these generalizations during the decision phases.

The point of learning is to summarize the characteristics of specific sounds from several speakers and speaking styles. If the descriptors--acoustic cues--carry enough discrimination power and the examples cover enough of the spectrum chosen, the learning system will produce recognition rules whereby later, unidentified acoustic descriptions can be identified as instances of specific classes of sounds.

Full discrimination might not be possible, either because the examples chosen for learning don't present enough information, or because the very nature of the descriptors is insufficient to provide resolution. An example from another domain demonstrates how descriptors affect resolution: if the learning task involved the discrimination between bicycles and cars, and the training examples contained only information about their color, resolution would not be possible.

The possible presence of errors in the data naturally presents further difficulties for discrimination. Unless they are recognized as such, bad data points can contradict and restrict generalizations that might be perfectly valid. Even when promising characterizations are made via the learning system, they may be refuted during experimentation, particularly when

new speaking styles are observed: they may now represent overgeneralizations.

Thus the aim of the learning system is to give descriptions of classes that are as general as possible given the data, and to also give characterizations of those areas where discrimination is confused. To accomplish this as effectively as possible in a real world environment, it must also try to identify and deal appropriately with errors in the data.

Faced with results that provide only partial discrimination for some classes, the teacher (in this case the expert developing the ASR) can either direct further learning by the program to amend the characterizations, either widening or narrowing them, or he can use the description of the confusion areas (i.e. any specifically delineated confusion between classes, or the areas not covered by the class generalizations) as the precondition(s) to further elaboration. That is, the teacher can present more detail aimed at clarifying some aspect of the generalizations, or if such actions appear unpromising, the teacher can decide to use what generalizations exist and to use the characterizations of the confusions to trigger a new step of elaboration. A new elaboration would, as mentioned, be created by the expert with the aid of the GPES. Naturally, for such elaboration, new information (most likely from new descriptors chosen specifically to resolve the confusion) should be presented to the learning system.

Since the teacher can decide to augment what has been learned with more examples, learning should be conducted incrementally. This implies that the learning program should be capable of saving and restoring its current state (learned generalizations), so that it may continue learning without having to reprocess all the data it had previously used.

To illustrate by analogy, the task of developing the ASR with respect to learning may be viewed as one of recognizing outlines on a badly focussed photograph. If one imagines a painting depicting several areas of differing colors that have distinct boundaries, this can represent the problem space, where the different color areas represent the different classes and characterization involves describing the boundaries on each area. Consider a poorly

focused photograph taken of the painting so that the image portrays the painting with general areas that are clearly of a certain color but whose edges meld in with the neighbouring color areas. This photograph can represent the learning space at some point in the development of the system. Lastly, if one now imagines such a photograph as emerging into view gradually, pixel by pixel, and if one considers that some pixels may be out of place, one creates a mental image of the task of learning in this environment.

The poor focus, lending an image of areas of indeterminate color, is meant to represent the problem of inadequate discriminant power of the descriptors. The image emerging piece by piece and never in full detail is meant to convey the incremental way in which the data is presented to the learning system. The picture is never complete, but more pieces of it can always be additionally revealed--just as more examples can be added to an incremental learning system for better informed descriptions--to give a more detailed picture. Indeed, whole areas of color may have been omitted from view and later brought to light, just as new speaking styles for the same sounds can be later observed. Lastly, in describing the analogy, the idea of misplaced pixels represents erroneous data or noise.

Although the picture is incomplete and badly focussed, some areas might still be perceived as quite recognizably belonging to one distinct color. The true boundaries might not be distinguishable, but an area within them can reasonably be described as a characterization for a given color. The true boundaries might be guessed at. Such descriptions are only partially discriminant; Although some of the problem space is now labelled, there still exist fuzzy or confusion areas.

The problem of erroneous data, here in the form of misplaced pixels, can be dealt with in the same manner as human perception deals with such situations; if enough of the picture emerges, a single lone point of one color sitting amongst a multitude of others of another color becomes more and more obviously contrasting as the picture emerges and must eventu-

ally be dismissed as erroneous from overwhelming evidence and then ignored. If, however, there are others of the same or different colors nearby to support it, the situation can change, where the contrasting color points might represent a local fuzzy area. On more evidence, the contrasting points might again be simply considered erroneous data. Thus the descriptions and hypotheses of the picture can change many times as the image gradually appears.

To extend the analogy, the teacher upon being given the learned outlines of the colored areas now faces two choices; to add more detail to the picture or to take another photograph, perhaps with another focus detailing some fuzzy area. These options correspond to more learning and to elaboration in the ASR described here.

As mentioned above, the data in a knowledge based approach consists of acoustic cues. Leaving out the descriptions of what is being measured in ASR and discussions on particular relevance, we restrict our study of the data to representation issues in learning. The acoustic cues are measures of the input signal that record either the simple absence or presence of some feature or some numerical value associated with a specific instance of some feature present in the signal. As an example of each type, we have for instance the cue 'Detection of a Buzz' (abbreviated 'bz') as a type of cue that provides description of the signal by its presence or absence, and 'Maximum Zero Crossing Density of the Signal' (abbreviated 'zx') which with some specific numerical count describes a signal feature.

Thus the language needed to describe the instances of examples for learning is in fact quite simple. All that is necessary is a set of attribute-value pairs to describe the features and the measures made. Those cues whose information consists simply of the presence or absence of some feature can also be represented as attribute-value pairs whose range is [0,1]. As illustration, an example of the data used to test the system looks as follows:

```
(observe '((buzz = 0.099) (resener = 0.092) (npeak = 0.500) (delay = 1.000)
        (bzival = 0.117) (mindex = 0.700) (mprofile = 0.021)
        (cg_index = 0.800) (index70 = 0.667) (bzcnt = 0.400) (bzmag = 0.336)
        (compactness = 0.680)) 'P)
```

where P is the class that this data is a positive example of, and buzz, resener, npeak, and delay etc. are the cues.

This consistency in representation permits a simple and uniform method of generalization accomplished by the application of a single rule. As all descriptors have as domain a linear type, the closing interval generalization rule as a special case of the adding alternative rule (see Section 2.4.3.1) is used exclusively to create generalizations.

A full learning example is represented as a conjunction of all its linear descriptors. However, as each example represents a single point in the problem space, possibly an erroneous one, and since the description, although consisting of several conjunctively linked cues, is an interpretation of a single event, each example is taken as an indivisible whole and generalization between them involves all the descriptors. That is, no intra-operator search is performed to determine which features get generalized. For example, we might have:

(a = 1) (b = 4) (c = 2)
(a = 0) (b = 7) (c = 6) ⇒ (a = 0..1) (b = 4..7) (c = 2..6)

where the conjunctions between the descriptors in the observations on the left and in the generalization on the right are implicit. Note that this is not quite how the closing interval generalization rule is expressed, where generalization occurs based on the disparity between observations in only one descriptor (all others being equal). The form of generalization used here is in fact wider and, although more prone to produce overgeneralizations, when coupled with backtracking methods and enough data, the scheme can produce valid generalizations. So in addition to the uniformity of representation, the use of each description as an indivisible whole simplifies the generalization task even more by eliminating the need for search through the combinatorially explosive sub-space within descriptions.

Moreover, the domain of the descriptors is known, and can be supplied to the learning program. Knowledge of the the domain allows the program to deal with examples that are presented with missing descriptors. The absence of a descriptor is meant to convey 'don't care' situations where the value associated with the missing descriptor was not observed and is presumably not of any consequence to the observation. In such cases, the program can provide the missing descriptor for the new observation as a generalization covering its entire domain. For example, if the domain of the descriptor 'a' is known to be [1,5], and the observation (b = 2) (c = 3) is presented to the learning system, then the program can translate this observation into (a = 1..5) (b = 2) (c = 3). Thus, by expressing that the 'missing' descriptor can take on any value in its domain, the 'don't care' message is preserved and the data is put into a standardized form for easier and quicker processing. The reverse operation, that of dropping descriptors (whose interval of values span the entire range) from generalized descriptions can easily be accomplished as a postprocessing operation in order to express the learned generalizations as simply as possible.

The uniformity of descriptions as ordered sequences of linear variables permits a view of them as n-tuples describing points (or areas, or spaces) in n-space. Indeed this spatial representation of the learning task provides a good illustration of its nature. Labelled points appear one by one in the space, and through generalization partitions of that space are described as belonging to the different classes. Some partition belonging to a class may be found to be so encompassing as to include too many points that are labelled differently from the class the partition represents, and its boundaries have to be redefined by using fewer, or perhaps different points of its own class. A two dimensional case is sufficient for illustration (fig. 3). The graph depicts the learned areas (solid squares) for two classes (circles and triangles). The areas are generalizations obtained from the individual points supplied as examples of each class (depicted according to membership). In the example, the triangle class descrip-

tion is too general because it includes too many (three in this case) observations from a different class. The triangle class description is therefore made less general (only extending as far right as the dotted line) by ignoring two observations of its own class, in order to remain consistent and within tolerances for error.

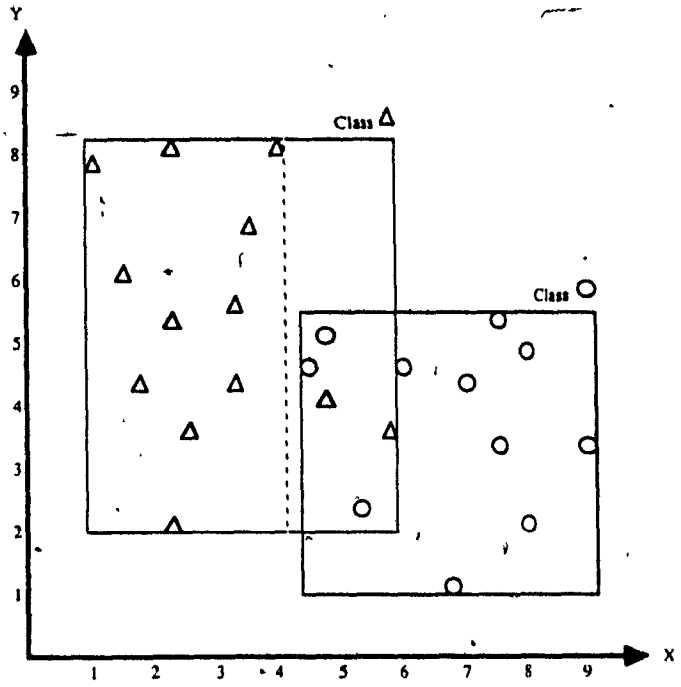

Figure 3: Graphical depiction of generalization.

One aspect of the problem well illustrated by this representation is the result of insufficient information carried by the examples for full discrimination. If enough information were present in the data to fully discriminate the concepts chosen for learning, assuming this is possible and discounting errors, the resulting generalizations would include all points of their class and the boundaries between them would be completely well-defined. This is to say that there would exist no overlaps forcing redefinition of the generalized boundaries and thus leaving some points outside. Given this situation, where enough information is present for full discrimination, if one now removes one or more dimensions, i.e. provides less information in

describing data, then the projections of the learned ideal generalizations into the lesser space might now show overlaps. The concepts may well be distinct, but with only partial information, only partial discrimination may be possible (fig. 4).

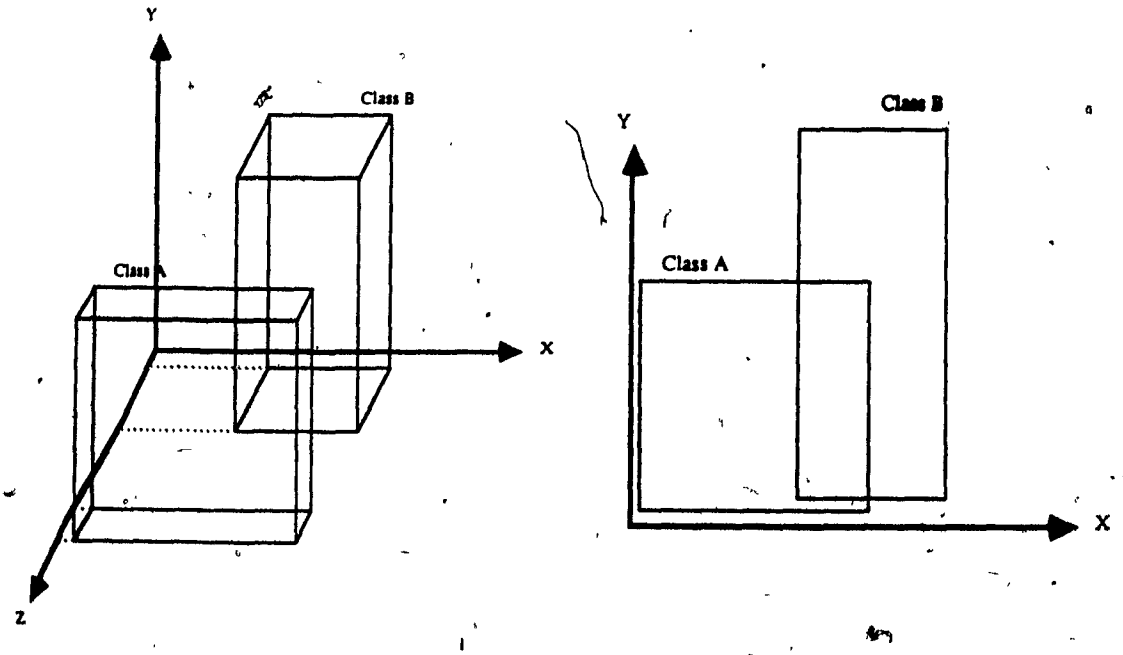

Figure 4: Overlap caused by loss of dimension.

3.3 Sources of noise.

Data collected during development of the ASR must be collected from the real world. Such data is never perfect. Therefore the system must be capable of dealing with the problems of noise found therein.

Data is collected by tape recording a series of individual speakers pronouncing sounds (letters, numbers, words etc.) by reading a list chosen to address the class of sounds for which a recognition plan is being realized. The recordings are then analysed and measured to produce the desired descriptions for learning. This same method is used when the ASR is

being tested for its recognition capacity, however during learning for development of the system, the class is supplied with each description. During testing, the system must determine the class if it can.

The descriptions obtained by analysing the signal represent real events and not interpretations, therefore any errors and anomalies existed before the analysis. Such errors therefore result from the speaker, the limitations of the medium used to record the data, and the method of collecting the data.

As was noted, the approach to developing the ASR is evolutionary. Perception and learning can not be expected to accomplish the entire task in one shot. Certain classes are targeted and efforts are focussed on their resolution. Along with a restriction of class during development, the speaking style must also be restricted. Although the aim of learning is to generalize for the class, the features chosen for extraction might only be capable of sufficiently characterizing sounds spoken in a certain style. Therefore, at a given stage in development, the speaker from whom data is being collected is not only restricted in what is said but in how it is said as well. The range of the style might be quite narrow for some perception plans. An utterance, although quite distinguishable to the human ear as belonging to a certain class, may be quite confusing to the learning system. A poorly spoken example (not spoken in a style that is being targeted for recognition) constitutes a 'mispronunciation'. This type of 'mispronunciation' represents one of the sources of noise. Outright mispronunciations, that is incorrect readings, also contribute to noise.

Another possible source of error is the simple mislabelling and bad placement or selection of data and other such 'clerical' errors that are possible in any large data collection project. Although these are usually the types of errors that can be screened out, they can be dealt with in the same way as less tractable errors.

Lastly, noise can be in the form of real noise, as the word is most familiarly understood. Stray sounds, background noises, and anomalous sounds produced from the difficulties in bio-mechanical, mechanical and electronic conveyance of sound might find their way onto the tape and therefore into some descriptions.

3.4 Summary.

The aspects of a learning system demanded by the needs of the ASR described here involve the provision of discriminant descriptions for several classes of sounds from generalizations of the descriptions of example sounds. These example sounds are relatively simply expressed. Because of the possibly incomplete nature of the data, and with a view to the elaboration-decision paradigm used by the recognition system and its evolutionary development, only partially discriminant descriptions might suffice for the construction of the decision rules of a given cycle. As well, descriptions of confusion areas detailing the boundaries and the classes involved is also desirable information.

Because the system is designed to grow, gradually assimilating more classes and styles to recognize, and since the learning program interacts with an expert who can direct more learning, adding to previously observed data to produce more informed results, the program must be incremental in approach. It must be capable of saving its current configuration and restoring itself for any state.

The existence of noise in the data implies that the program should try to recognize errors and have some mechanism for ignoring them.

Lastly as it is meant as a practical tool, the learning system should accomplish these goals with reasonable speed and efficacy.

Chapter 4.

Previous Work.

4.1. Introduction.

The ASR by DeMori et al. [DeMori1987] has already benefited from an inductive learning program; designed by Michel Gilloux, it also was designed specifically for use in this ASR system. Unfortunately, for reasons described below, its operation was found to be inadequate. The present thesis is an effort to rectify the problems encountered by Gilloux's system. Thus, the present work has been modelled on Gilloux's work and has evolved from it. This chapter will briefly describe Gilloux's work and the problems found therein, thus providing an introduction to the work addressed in this thesis.

4.2. Structure of previous work

The structure of Gilloux's inductive learning program consists of a network of nodes constructed dynamically as the program assimilates data. Each node represents an observation or a generalization of two or more observations for a given class. The nodes contain the description and class of the observation or generalization. In accordance with the nomenclature for the task of generating recognition rules, Gilloux calls these the left-hand-side (LHS) and right-hand-side (RHS) of a node respectively. As the system is intended to operate with non-monotonic logic, a status field (values: 'in' or 'out') indicating current belief or disbelief of validity of the description is included with each node.

The nodes are linked via pointers that indicate relative generality. These links are kept in lists called 'IN' and 'OUT' and are attached to each node. The IN list carries the pointers that refer to all nodes of the same class as the host node and less general. The OUT list points to less general nodes of other classes. A node is less general than another if its description is completely contained within the description of the other. Thus the IN and

OUT lists act as the support lists of a TMS that provide the justification for a node's status. A node is 'in' if all the nodes on its IN list are also 'in'. This translates as: A generalization is believed only if all the lesser generalizations and observations on which it is based are also believed. Also, a node is 'in' only if all the nodes on its OUT list are 'out'. That is, it is believed only if all the lesser generalizations and observations of other classes that are encompassed by its description are disbelieved.

These links to less general nodes are doubled by providing the opposite link in the less general node to the more general node. Each node then also has IS-IN-OF and IS-OUT-OF lists that point to encompassing generalizations of the same class and of other classes. This structure provides a direct retrieval method for generalizations without necessitating search.

In order to capture the possibly imprecise nature of the data and to provide a framework for the data's expression in the formation of generalizations, each node is also supplied with positive and negative evidence measures. For clarity, these will be called 'evidence' and 'conflict'. Each node has a measure of evidence indicating the number of times it has been observed by the program. In the case of generalizations, the evidence measure consists of the sum of the evidence measures for the observations from which it is generalized. The conflict measure for an observation is always zero; that for a generalization is the sum of the evidence measures of all the observations that are less general and represent examples from classes other than the generalization's own class.

With respect to the evidence and conflict measures, the validity of a generalization can also be determined according to the weights of support and opposition to a description, regardless of the current status of the nodes involved. As implemented, a node is believed only if its conflict measure is below a certain threshold value and its evidence measure greatly outweighs its conflict measure in terms of some predetermined threshold ratio. As long as members of the IN and OUT lists are in accordance, a node can be believed even though

there is observed evidence that refutes the generalization, if that conflict is small enough both absolutely and relatively. Alternately, a node may be disbelieved based on these same criteria even though its current TMS state allows for its belief.

By this method, incomplete confidence in the data in terms of errors and in discrimination power can be accounted for in the generalization process. Belief can be accorded to generalizations even though there is some (possibly erroneous) small amount of refuting evidence. Disbelief can be accorded to generalizations whose nodes in opposition are all disbelieved, but whose validity is suspect because of the sheer volume of the opposition or because of its unacceptable ratio to the support.

Aside from the node structure, the only other data structure found in Gilloux's program is the organization of the network itself, which consists simply of a list of all the nodes placed in reverse order of creation.

4.3. Operation of previous work

In accordance with the nature of the task as outlined in chapter 3, Gilloux has designed an incremental inductive learning system. The network grows and reaches a new state as each observation is presented to it. The generalization method is that described in the previous chapter, whereby two descriptions of the same class are generalized to form one by creating intervals from the values for the matching descriptors.

As they are successively presented to the program, observations are first screened for previous occurrence. If the same observation (same RHS and LHS) already exists in the network, its evidence is simply increased. If the observation is new, it is introduced into the network as a new node and is compared to the rest of the nodes in the network in order to establish links and update support and opposition measures (Figures 5 and 6).

```
Procedure LEARNEXAMPLE (description, class)
  newnode := member(description, class, list-of-nodes)
  if node ≠ nil then
    begin
      evidence(node) := evidence(node) + 1;
      for every node N in is-in-of(node) do
        UPEVIDENCE(node);
      for every node N in is-out-of(node) do
        UPCONFLICT(node);
    end
  else
    ADDNODE(makenode(description, class));
end;
```

Figure 5: Gilloux's LEARNEXAMPLE procedure.

```
Procedure ADDNODE(node);
  for every node N in list-of-nodes do   (*list-of-nodes is all nodes in the*)
    if RHS(N) = RHS(node) then           (*network, globally defined.      *)
      begin
        if moregeneralthan(LHS(N), LHS(node)) then
          makeinlink(N,node)
        else
          if moregeneralthan(LHS(node), LHS(N)) then
            makeinlink(node,N)
          else
            if (not (member(generalize(LHS(node), LHS(N)), list-of-nodes))) do
              ADDNODE(makenode(generalize(LHS(node), LHS(N))));
      end

    else (* RHS(N) ≠ RHS(node) *)
      if equivalent(LHS(N), LHS(node)) then
        begin
          makeoutlink(N,node);
          makeoutlink(node,N);
        end
      else
        if moregeneralthan(LHS(N), LHS(node)) then
          makeoutlink(N,node)
        else
          if moregeneralthan(LHS(node), LHS(N)) then
            makeoutlink(node,N);
end;
```

Figure 6: Gilloux's ADDNODE procedure.

Both of these actions, since they represent new information, can cause a change in status for other nodes; either by being on the OUT list of some node from another class causing it to be disbelieved (observations are assumed to be 'in'), or by providing enough evidence to some previously disbelieved node allowing its rebelief (or vice-versa through conflicting evidence). When a node's status is changed, inconsistencies in the set of beliefs of the network can arise. For example, if a node N changes from 'in' to 'out' in status, other more general nodes from the same class ought also have their status changed to 'out'. Via the TMS formalism, the consequences of any change in status for a given node are traced to all the other nodes dependent on its status. Necessary changes are effected, and their own consequences are further propagated through the network. In this way, a consistent set of beliefs is maintained at all times (Figure 7).

Note also that when a node is disbelieved, it is retained in the network. So that if the evidences and the set of beliefs should later allow, it can be rebelieved without rederivation.

When a new node is being added to the network, it is generalized with all other nodes of the same class (both observations and generalizations), and when this results in previously unseen descriptions, the new generalizations are added to the network in exactly the same way. Naturally this can create further changes and more generalizations (Figure 6).

In this way, Gilloux's program builds up a network of nodes whose current state contains every generalization (believed or disbelieved) that can be made with the given observations, and it does so for each class. The current set of beliefs is based on the competing evidence of the observations presented to it. The widest of the believed nodes, (that is the best generalizations) can be traced as those that are currently 'in' and who are not linked to any others of the same class that are more general. There can be many of these and since the program covers all the generalizations, it is capable of learning disjunctive concepts; that is,

```
Procedure TRUTHMAINTAIN(node,status);
 if status = 'in' then
  begin
    status(node) := 'in';
    for every node N in is-out-of(node) do
     TRUTHMAINTAIN(N,out);
    for every node N in is-in-of(node) do
     if status(N) = 'out' and ADMISSIBLE(N) then
      TRUTHMAINTAIN(N,in);
  end

 else (* status is out *)
  begin
    status(node) := 'out';
    for every node N in is-in-of(node) do
     TRUTHMAINTAIN(N,out);
    for every node N in is-out-of(node) do
     if status(N) = 'out' and ADMISSIBLE(N) then
      TRUTHMAINTAIN(N,in);
  end;
end;
```

Figure 7: Gilloux's TRUTHMAINTAIN procedure.

offering several descriptions disjunctively linked.

4.4. Problems with previous work

Unfortunately, Gilloux's program fails as a workable solution in two respects. The first of these problems is that the exhaustive search for valid generalizations suffers from a combinatorial explosion. The network's size grows much too rapidly. Since each new node is generalized with every node of the same class and these generalizations are introduced as new nodes in the same way, the program in effect creates all possible generalizations for a class by generalizing the observations in every combination of all possible sizes. Although redundant generalizations are not copied in the network, all combinations are in fact generated, and the set of unique generalizations retained is still explosive in its growth.

Each new node must be compared with the rest, of all classes, to establish links, update the measures and maintain consistency. The processing of new observations quickly becomes more and more cumbersome and the time of processing new observations increases dramatically. This makes the program unusable for large amounts of data. During tests, evidence showed that the number of nodes in the network increased at a rate approaching 2^n where n was the number of observations made.

The second problem with Gilloux's program is that no mechanism is provided for deciding the validity of an observation. Although there is in place a method (by competing evidence and conflict) for taking the unreliability of the data into account when deciding the validity of generalizations, the strict adherence to the TMS formalism as a means of propagating changes and maintaining consistency coupled with the unwavering belief in observations eclipses the method's effects entirely.

An observation, a point in the learning space, acting as a node in Gilloux's network has no less-general nodes pointing to it and therefore has no members on its outlist and no conflict. Since it is also assumed to be 'in' initially, it remains 'in' for the duration of the network, as nothing can make it go 'out'. It follows from this situation that any generalization that goes 'out' will also remain so for the duration of the network: If a generalization goes 'out', it was compelled to do so because one or more opposing observations are 'in'. Although a node may be put 'out' because of an IN list member being 'out', or because of evidence and conflict measures, the reason for its disbelief can always be reduced to opposing observations being 'in'. That is, any given 'out' generalization can have the cause of its disbelief traced back ultimately to one or more opposing observations being 'in'. This includes nodes which have been made 'out' because of overwhelming conflict measures that represent several opposing observations.

Because of this flaw, Gilloux's program actually accomplishes the simpler task of finding

the generalizations of each class for which there is no conflict. The retention of out nodes, the use of T.M.S., the links and measures etc., all become ineffective as no node with any opposition is ever believed and once 'out', no nodes are ever rebelieved.

Chapter 5.

The learning program.

5.1. Introduction.

We propose here to accomplish the learning task as set out in chapter three, and also to address the problems encountered with Gilloux's solution as described in the previous chapter. The problem space and goals remaining the same, many characteristics of our proposed approach remain the same as Gilloux's: the representation issues and method of generalization remain unchanged, processing is still incremental, the direction toward solution is still specific-to-general and the program retains a data-driven strategy. We have tried to reduce the size of the network and increase its speed. In order to have error tolerance without interference from the truth maintenance mechanism, we made some changes in the way observations are perceived by the system and in the way changes in validity are propagated. This program then differs in structure, search control, and in the treatment of observations. The program retains a TMS-like manager to maintain consistency, but the program's operation differs in the new algorithm.

More organization has been imposed on the network structure. In order to facilitate the implementation of all aspects of the new algorithm, the network has evolved from Gilloux's simple list to a collection of nodes divided by class, and within each class by separating observations and generalizations.

The most significant difference in approach to converging on a generalization that characterizes a class is the control strategy of the search. A depth-first approach was adopted for this program, where the aim was to develop and retain at any given point in the execution of the learning program only the widest (i.e. most general) or few widest generalizations that are believed valid for each class in the network. No intermediary generalizations are retained. If a generalization becomes too wide, the program backtracks to a different or

lesser generalization that is consistent with the state of the other nodes. This approach is heuristic because the program does not generate all the generalizations possible and will not abandon the current widest generalizations to try combinations of observations that differ greatly unless compelled to backtrack far enough.

For this reason the 'best' generalizations might not ever get generated. However, as with most heuristic solutions to problems in artificial intelligence, the goal of finding the best solution is sacrificed so that an adequate solution might be found more conveniently.

To compare, Gilloux's program tries to find a solution by a brute force strategy, expending most of its efforts creating all possible generalizations. Our program searches for a solution in a more directed manner, trying to find another possible generalization only when necessary for consistency.

In Gilloux's program, all the mechanisms necessary to implement a method of determining validity of generalizations with error tolerance were in place, but were made ineffective because of the absolute validity accorded to all observations. In order to rectify this situation, a criterion by which observations can be deemed invalid is introduced. In the new program, an observation is assumed valid initially, but it retains this status only when it is currently contributing to the (widest) generalizations believed valid for its class. That is, an observation is believed only if it is a member of the IN list of some currently held generalization of the same class. An observation is disbelieved ('out') otherwise. That is, the observation is not accorded any validity if it is not among those observations that are generalized to form the current best generalizations.

This method allows the validity of generalizations to be determined by the competing evidence and conflict measures, as was the intent of Gilloux's work. The purpose of the TMS is simply to propagate any changes determined in this way in order that consistency be maintained. With this criterion for observations, these intentions are better realized and a

more dynamic network is achieved.

5.2 Structure of the learning program.

As mentioned above, an organized structure has been imposed on the network in order to improve programming, legibility and comprehension, as well as processing.

The network is divided into class groups. A class group is created dynamically for each new class presented to the program. Each class group is a list containing the name of the class followed by 4 sublists containing the nodes (observations and generalizations) belonging to that class. They are in order:

1. Observations: The observed nodes: the nodes that result from direct inputs to the program as learning examples.

2. Generalizations: The current most general nodes that are believed.

3. Gensout: The generalized nodes that are disbelieved but whose status may change dependent on the belief accorded to other nodes.

4. Transitions: The list of observation nodes whose status is pending during a propagation of changes through the network.

This last list is necessary for consistency maintenance. It is only non-empty during change propagations and empty whenever the network is "up-to-date". Its contents do not represent any additional information. Its use is made more apparent in the discussion of propagation below.

The structure of a node is much the same as it was in Gilloux's network. The distinction made between observations and generalizations and the listing conventions described below for the IN and OUT lists eliminates the need for the reverse link IS-IN-OF and IS-OUT-OF lists. The only other difference is in the naming conventions for the IN and OUT lists, which call 'proponents' and 'opponents' to reflect their role better.

A generalization is completely characterized by its proponents list. Every observation of the same class that is less general is included in it. Any generalization of the same class that is less general would be composed of a subset of these proponents and would therefore be redundant if it were also included on the proponents list of the more general node. By restricting links to exist only between observations and generalizations, comparisons about relative generality and similarity between generalizations of the same class are simplified: if one generalization's proponents list contains all the members of another's, then it is more general than (or possibly equal to) the other.

Thus, the list of nodes in the 'proponents' of a generalization consists only of the observations from which it is derived. The list of 'opponents' is the list of observations (only) from other classes that fall within the span of its description. On the other side, an observation's 'proponents' list contains all the existing generalizations that are built from it, and its 'opponents' list contains all the generalizations of other classes that contain it; that is, those it supports and those it opposes.

As always, generalizations are 'in' according to the rules of TMS (all opponents 'out', all proponents 'in'). The status of a generalization also depends on the formula for error tolerance involving evidence and conflict measures.

5.3 Description of processing in the learning program

We now describe the program in detail.

5.3.1 Observation.

One by one, learning examples for different classes are presented to the program. Each example consists of a description and a class (Figure 8). Optionally a number can be provided indicating how many observations a given example represents. The descriptors are put

into a standard order for convenience of comparison and generalization. This includes filling in any 'don't care' missing descriptors according to an alphabet that expresses the domain of each as described on page 40 above. The alphabet is optionally incorporated into the program. If it does not exist, the learning data should be presented in a consistent format for proper execution.

```
Procedure OBSERVE(description, class, evidence);
(* Introduces new examples into the network. Searches
   within the same class and within the other classes
   for previous observation or contradiction. *)

begin
 align description in standard form;
 if class does not exist then
  create-structure(class);

 search observations-of(class) for same description;
 if previous-observation then
  UPEVIDENCE(previous-observation, evidence)

 else
  begin
   search observations-of all other classes
      for same description;
   if contradicting-node then
    begin
     evidence:= evidence + evidence-of(contradicting-node);
     class:= union(class, class-of(contradicting-node);
     KILL(contradicting-node);
    end;

   newnode:= create-node(description, class, evidence);
   addnode(newnode);
  end;
end;
```

Figure 8: Pseudo-code for OBSERVE.

If a class structure does not already exist for the given class, one is created to hold the nodes of that class.

The new description is compared to those of previous observations in the same class to discover if the new example has already been encoded. If so, its evidence is increased as is the evidence of those generalizations it proposes and the conflict measures of those it opposes (Figure 9). The consequences of these changes in measures can alter the beliefs ('in' to 'out' or vice versa) accorded some generalizations. The details of those events are found below under Believe and Disbelieve.

```
Procedure UPEVIDENCE(node, evidence);
(* Increases the evidence of an observed example
   and effects the changes influenced by this. *)

Var believelist: list-of-nodes;

begin
 evidence-of(node):= evidence-of(node) + evidence;

 for every P in the proponents of node do
  begin
   evidence-of(P):= evidence-of(P) + evidence;
   if (out-p(P)) and (admissible-p(P)) then
    push(P, believelist);
  end;

 for every opponent O in the opponents of node do
  begin
   conflict-of(O):= conflict-of(O) + evidence;
   if (in-p(O)) and not(admissible-p(O)) then
    DISBELIEVE(O);
  end;

 BELIEVE(believelist); (* a list of candidate nodes for belief *)
                       (* test, change status, and propagate if valid *)
 while not empty(changestack) do
  PROPAGATE(pop(changestack));
end;
```

Figure 9: Pseudo-code for UPEVIDENCE.

If the example is new, the description is compared against all observations of other classes. If an observation from a different class is found with the same description, a con-

tradiction is detected and resolved as described in the next chapter.

Having passed these tests, the example is entered into the network as an observation node (Figure 10).

Entering an example in the network is accomplished by adding it to the list of observations for its class, and then linking it to those generalizations of its own and other classes that it is less-general than. This insertion and linkeage also involves updating the evidence or conflict measures of those it now supports or opposes. This action in turn can change the validity of those nodes (as tested here by the function admissible-p) from 'out' to 'in' in the case of 'out' generalizations of its own class, and 'in' to 'out' for 'in' generalizations of other classes.

Following this, the observation is merged with the currently believed widest generalizations of its own class to create new generalizations. If any of these are admissible, they are retained as the new, more general generalizations for the class; generalization also involves determining proponents and opponents as well as evidence and conflict measures for the newly synthesized description. If, however, these generalizations do not result in any new descriptions that are acceptable under the present circumstances, the observation is deemed 'out', and the previous generalizations are retained. All the changes accumulated by these operations are then propagated.

In this way, the program only merges the new observations with the most general generalizations already held. Only a small number of these widest generalizations are held in the network at any given time. The number of these determines how many branches of the search tree need be investigated, and this number is regulated by a heuristic evaluation described in Section 5.3.2. The merging of observations only with the best generalizations attained yet implements a depth-first-like strategy for search. Any lesser intermediary generalizations have to be generated during backtracking, when none of the generalizations

Of course, the two methods differ radically in operation and how learning is achieved. The present uses logic with inductive rules to manipulate descriptions and create new generalizations. Neural networks depend on gradual numerical adjustments in parameters to create recognition systems. Both approaches have their advantages. In fact, recent advances in the field of neural networks have proven very useful to the task of speech recognition. The main advantage to using a learning system based on a more formal logic (when contrasted to neural nets) is the ability to express some problem background knowledge to help guide the generalization process in the system. Neural networks are more limited in this respect.

To achieve a speech recognition system that approaches human performance levels is a very difficult task. For this reason the current developments in the ASR involve learning from several sources including stochastic methods and neural networks to make best use of the advantages each has to offer. The suggested role for an inductive learning program in the emerging system is as a 'filter' for learning data. The present program organizes the data so that a partition of the learning space is produced whereby individual examples can be judged according to how well they represent their class or how problematic they are (through inclusion in confusion areas or as probable errors.) The other learning systems might benefit from such a filter in the presentation of data as they do not have these mechanisms for determining the relative worth of training examples. Thus this system would retain its function as a guide to development and take on the new role of teaching aid.

Apart from the role for which it was intended, suggestions have been made for application of the system in the domain of financial classification. Models in this discipline exist whose purpose is to partition a space of variables for predictive purposes (most notably the prediction of financial distress) [Frydman1985]. The task of learning in this area includes the possibility of mislabeled learning examples, thus the methods presented in this thesis seem very well matched to the task of learning the intervals describing such partitions.

7.5 Suggested improvements.

There is room for improvement of this system in many areas. Most notably, more sophisticated search techniques could be used in the quest for the best generalization. The method by which errors are found could be improved to distinguish them from observations that are simply part of irresolute areas.

Both of these goals can be achieved by adding more domain specific knowledge to the system. Expectations for the data could be expressed and used to screen out likely errors. Certainly searches can be aided by more informed evaluations.

Lastly, entirely different learning methodologies such as explanation-based learning [DeJong1981] might be applied to the task required for the ASR. Such investigations provide fruitful areas for further research.

References

Angluin1983.
D. Angluin and C. Smith, "Inductive Inference: Theory and Methods," *ACM Computing Surveys*, vol. 15, no. 3, pp. 237-269, September 1983.

Carbonell1983.
J.G. Carbonell, R.S. Michalski, and T.M. Mitchell, "An Overview of Machine Learning," in *Machine Learning: An Artificial Intelligence Approach*, pp. 3-24, Tioga, Palo Alto CA., 1983.

Carbonell1983a.
J.G. Carbonell, "Learning by Analogy: Formulating and Generalizing Plans from Past Experience," in *Machine Learning: An Artificial Intelligence Approach*, eds. J.G. Carbonell, R.S. Michalski, and T.M. Mitchell, pp. 137-161, Tioga, Palo Alto CA., 1983a.

Caudill1988.
M. Caudill, "Neural Networks Primer," *AI Expert*, pp. 46-52, 55-61, December 1987, February 1988.

Charniak1985.
E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*, pp. 609-659, Addison-Wesley, Reading, Mass., 1985.

DeJong1981.
G. DeJong, "Generalizations Based on Explanations," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1981.

DeMori1983.
R. DeMori, *Computer Models of Speech Using Fuzzy Algorithms*, Plenum Press, New York, 1983.

DeMori1987.
R. DeMori, L. Lam, and M. Gilloux, "Learning and Plan Refinement in a Knowledge-based System for Automatic Speech Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, pp. 289-305, March 1987.

Dietterich1982.
T. Dietterich, B. London, K. Clarkson, and G. Dromey, "Learning and Inductive Inference," in *The Handbook of Artificial Intelligence*, eds. A. Barr, P.R. Cohen, and E.A. Feigenbaum, vol. 3, pp. 325-511, HeurisTech press, Stanford, Cal., 1982.

Dietterich1983.
T.G. Dietterich and R.S. Michalski, "A comparative Review of Selected Methods of Learning from Examples," in *Machine Learning: An Artificial Intelligence Approach*, eds. J.G. Carbonell, R.S. Michalski, and T.M. Mitchell, pp. 41-81, Tioga, Palo Alto CA., 1983.

Doyle1979.
J. Doyle, "A Truth Maintenance System," *Artificial Intelligence*, vol. 12, no. 3, pp. 231-272, 1979.

Ellman1986.
T. Ellman, "Explanation-Based Learning in Logic Circuit Design," in *Machine Learning: A Guide to Current Research*, eds. T.M. Mitchell, J.G. Carbonell, and R.S. Michalski, pp. 63-66, Kluwer Academic Press, Boston, Mass., 1986.

Foderado1983.
J.K. Foderado, K.L. Sklower, and K. Layer, *The Franz Lisp Manual*, The Regents of the University of California, 1983.

Frydman1985.
H. Frydman, E.I. Altman, and D. Kao, "Introducing Recursive Partitioning for Financial Classification: The Case of Financial Distress," *The Journal of Finance*, vol. XL, no. 1, pp. 269-291, March 1985.

Genesereth1987.
M.R. Genesereth and N.J. Nilsson, *Logical Foundations of Artificial Intelligence*, pp. 161-176, Morgan Kaufmann, Los Altos, Cal., 1987.

Hayes-Roth1983.
F. Hayes-Roth, "Using Proofs and Refutations to Learn from Experience," in *Machine Learning: An Artificial Intelligence Approach*, eds. J.G. Carbonell, R.S. Michalski, and T.M. Mitchell, pp. 221-240, Tioga, Palo Alto CA., 1983.

Hedrick1976.
C.L. Hedrick, "Learning Production Systems from Examples," *Artificial Intelligence*, vol. 7, no. 1, pp. 21-49, 1976.

Hood1986.
G. Hood, "Neural Modeling as One Approach to Machine Learning," in *Machine Learning: A Guide to Current Research*, eds. T.M. Mitchell, J.G. Carbonell, and R.S. Michalski, pp. 103-108, Kluwer Academic Press, Boston, Mass., 1986.

Kibler1983.
D. Kibler and B. Porter, "Perturbation: A Means of Guiding Generalization," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 415-418, William Kaufmann, 1983.

Kinber1977.
E.B. Kinber, "On a Theory of Inductive Inference," *Lecture Notes in Computer Science*, vol. 56, pp. 435-440, Springer-Verlag, 1977.

Langley1985.
P. Langley and T.M. Mitchell, "Machine Learning," *Ninth International Joint Conference on Artificial Intelligence Conference Tutorial Program Notes*, 1985.

Langley1987.
P. Langley, "A General Theory of Discrimination Learning," in *Production System Models of Learning and Development*, eds. D. Klahr, P. Langley, and R. Neches, pp. 99-161, The MIT Press, Cambridge, Mass., 1987.

Lenat1983.
D.B. Lenat, "The Role of Heuristics in Learning by Discovery: Three Case Studies," in *Machine Learning: An Artificial Intelligence Approach*, eds. J.G. Carbonell, R.S. Michalski, and T.M. Mitchell, pp. 243-306, Tioga, Palo Alto CA., 1983.

Michalski1980.
R.S. Michalski, "Pattern Recognition as Rule-guided Inductive Inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, no. 4, pp. 349-361, 1980.

Michalski1983.
R.S. Michalski, "A Theory and Methodology of Inductive Learning," in *Machine Learning: An Artificial Intelligence Approach*, eds. J.G. Carbonell, R.S. Michalski, and T.M. Mitchell, pp. 83-134, Tioga, Palo Alto CA., 1983.

Michalski1983a.
R.S. Michalski and R.E. Stepp, "Learning from Observation: Conceptual Clustering," in *Machine Learning: An Artificial Intelligence Approach*, eds. J.G. Carbonell, R.S.

Michalski, and T.M. Mitchell, pp. 331-363, Tioga, Palo Alto CA., 1983a.

Michalski1986.
R.S. Michalski, "Machine Learning Research in the Artificial Intelligence Laboratory at Illinois," in *Machine Learning: A Guide to Current Research*, eds. T.M. Mitchell, J.G. Carbonell, and R.S. Michalski, pp. 193-197, Kluwer Academic Press, Boston, Mass., 1986.

Mitchell1977.
T.M. Mitchell, "Version Spaces: A Candidate Elimination Approach to Rule Learning," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 305-310, Morgan Kaufmann, Los Altos, Cal., 1977.

Mitchell1982.
T.M. Mitchell, "Generalization as Search," *Artificial Intelligence*, vol. 18, no. 2, pp. 203-226, March 1982.

Mitchell1982a.
T.M. Mitchell and P.E. Utgoff, "Acquisition of Appropriate bias for Inductive Concept Learning," *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, pp. 414-417, Morgan Kaufmann, Los Altos, Cal., 1982a.

Nilsson1980.
N. Nilsson, *Principles of Artificial Intelligence*, Tioga, Palo Alto, Cal., 1980.

Quinlan1983.
J.R. Quinlan, "Learning Efficient Classification Procedures and their Application to Chess End Games," in *Machine Learning: An Artificial Intelligence Approach*, eds. J.G. Carbonell, R.S. Michalski, and T.M. Mitchell, pp. 463-482, Tioga, Palo Alto CA., 1983.

Rendell1986.
L. Rendell, "A scientific Approach to Practical Induction," in *Machine Learning: A Guide to Current Research*, eds. T.M. Mitchell, J.G. Carbonell, and R.S. Michalski, pp. 269-274, Kluwer Academic Press, Boston, Mass., 1986.

Rich1983.
E. Rich, *Artificial Intelligence*, McGraw-Hill, New York, 1983.

Riesbeck1981.
C.K. Riesbeck, "Failure-Driven Reminding for Incremental Learning," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, vol. 1, pp. 115-120, Morgan Kaufmann, Los Altos, Cal., 1981.

Segen1985.
J. Segen, "Learning Concept Descriptions from Examples with Errors," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 364-366, Morgan Kaufmann, 1985.

Segen1986.
J. Segen, "Learning from Data with Errors," in *Machine Learning: A Guide to Current Research*, eds. T.M. Mitchell, J.G. Carbonell, and R.S. Michalski, pp. 299-302, Kluwer Academic Press, Boston, Mass., 1986.

Simon1983.
H.A. Simon, "Why Should Machines Learn?," in *Machine Learning: An Artificial Intelligence Approach*, eds. J.G. Carbonell, R.S. Michalski, and T.M. Mitchell, pp. 25-38, Tioga, Palo Alto CA., 1983.

Smith1977.
R.G. Smith, T.M. Mitchell, R.A. Chestek, and B.G. Buchanan, "A Model for

Learning Systems," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 338-343, Morgan Kaufmann, 1977.

Steele1984.
G.L. Steele, *Common Lisp: The Language*, Digital Press, 1984.

Tadepalli1986.
P.V. Tadepalli, "Learning in Intractable Domains," in *Machine Learning: A Guide to Current Research*, eds. T.M. Mitchell, J.G. Carbonell, and R.S. Michalski, pp. 337-341, Kluwer Academic Press, Boston, Mass., 1986.

Valiant1985.
L. Valiant, "Learning Disjunctions of Conjunctions," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 560-566, Morgan Kaufmann, 1985.

Valiant1986.
L.G. Valiant, "What Can Be Learned?," in *Machine Learning: A Guide to Current Research*, eds. T.M. Mitchell, J.G. Carbonell, and R.S. Michalski, pp. 349-351, Kluwer Academic Press, Boston, Mass., 1986.

Vere1975.
S.A. Vere, "Induction of Concepts in the Predicate Calculus," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pp. 281-287, Morgan Kaufmann, Los Altos, Cal., 1975.

Whitehill1980.
S.B. Whitehill, "Self Correcting Generalization," *Proceedings of the First National Conference on Artificial Intelligence (AAAI-80)*, pp. 240-242, William Kaufmann, Los Altos, Cal., 1980.

Winston1975.
P.H. Winston, "Learning Structural Descriptions from Examples," in *The Psychology of Computer Vision*, eds. P.H. Winston, p. ch. 5, McGraw Hill, New York, 1975.

Winston1977.
P.H. Winston, *Artificial Intelligence*, Addison-Wesley, Reading, Mass., 1977.

Appendix A.

Some results for ASR

The following is an abbreviated report of the some learning achieved by this system when given data from the E1 set of letters and digits. It is provided as an example of the type of output produced.

Learning Report

Total number of nodes: 172
Total learning time: 69

Classes included: ((b p) (3) (e v) (b e) (4) (g) (k) (t) (v) (p) (b) (d) (e))

Direct contradictions in observations:
  class: (b p)
    predicate: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
          (dl = 2) (dr = 3) (np = 1) (pb = 1) (re = 0)
          (rq = 2) (rr = 2) (st = 0) (zq = 3))

  class: (e v)
    predicate: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
          (dl = 1) (dr = 4) (np = 0) (pb = 1) (re = 0)
          (rq = 2) (rr = 2) (st = 0) (zq = 3))

  class: (b e)
    predicate: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
          (dl = 2) (dr = 3) (np = 1) (pb = 1) (re = 0)
          (rq = 3) (rr = 2) (st = 0) (zq = 2))

Generalizations

Class: (b p)
  number of nodes observed: 1

  pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
      (dl = 2) (dr = 3) (np = 1) (pb = 1) (re = 0)
      (rq = 2) (rr = 2) (st = 0) (zq = 3))
  evidence: 3 conflict: 0
  opposing observations:

observations not included:

Class: (3)
number of nodes observed: 1

pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 1) (st = 0) (zq = 1))
evidence: 1 conflict: 0
opposing observations:

observations not included:

Class: (e v)
number of nodes observed: 1

pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 4) (np = 0) (pb = 1) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 3))
evidence: 2 conflict: 0
opposing observations:

observations not included:

Class: (b e)
number of nodes observed: 1

pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 3) (np = 1) (pb = 1) (re = 0)
(rq = 3) (rr = 2) (st = 0) (zq = 2))
evidence: 2 conflict: 0
opposing observations:

observations not included:

Class: (4)
number of nodes observed: 1

pred: ((bi = 3) (br = 2) (bu = 1) (bz = 1)
(dl = 1) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 3) (rr = 3) (st = 1) (zq = 3))
evidence: 1 conflict: 0

opposing observations:

observations not included:

Class: (g)
number of nodes observed: 3

pred: ((bi = 3) (br = (2 3)) (bu = 1) (bz = 1)
(dl = (1 2)) (dr = 3) (np = (0 1)) (pb = 0) (re = 0)
(rq = (2 4)) (rr = 2) (st = (1 2)) (zq = (1 2)))
evidence: 3 conflict: 0
opposing observations:

observations not included:

Class: (k)
number of nodes observed: 5

pred: ((bi = 1) (br = 2) (bu = (0 1)) (bz = (0 1))
(dl = (1 4)) (dr = (2 3)) (np = 1) (pb = (0 1)) (re = 0)
(rq = (3 4)) (rr = (2 3)) (st = (0 1)) (zq = 1))
evidence: 3 conflict: 1
opposing observations:
(d)--> ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 3) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = 1) (zq = 1))

pred: ((bi = 1) (br = 2) (bu = (0 1)) (bz = 0)
(dl = (1 2)) (dr = 3) (np = (1 2)) (pb = (0 1)) (re = 0)
(rq = 4) (rr = (1 2)) (st = (0 1)) (zq = 1))
evidence: 3 conflict: 1
opposing observations:
(3)--> ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 1) (st = 0) (zq = 1))

observations not included:
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 4) (np = 1) (pb = 1) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 2))

Class: (t)
number of nodes observed: 3

pred: ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 2) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 2) (rr = 3) (st = 1) (zq = 1))
evidence: 1 conflict: 0
opposing observations:

pred: ((bi = 1) (br = (2 3)) (bu = 0) (bz = 0)
(dl = 2) (dr = (2 4)) (np = 1) (pb = 1) (re = 0)
(rq = (3 4)) (rr = 1) (st = 0) (zq = (1 3)))
evidence: 2 conflict: 0
opposing observations:

observations not included:
pred: ((bi = 1) (br = 3) (bu = 0) (bz = 0)
(dl = 2) (dr = 4) (np = 1) (pb = 1) (re = 0)
(rq = 3) (rr = 1) (st = 0) (zq = 3))

Class: (v)
number of nodes observed: 12

pred: ((bi = (1 4)) (br = (3 4)) (bu = (0 1)) (bz = (0 1))
(dl = (1 2)) (dr = (3 4)) (np = (0 1)) (pb = 0) (re = (0 1))
(rq = (1 3)) (rr = (3 4)) (st = (1 2)) (zq = (1 4)))
evidence: 8 conflict: 1
opposing observations:
(b)--> ((bi = 2) (br = 4) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 3) (st = 1) (zq = 3))

observations not included:
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 1) (pb = 0) (re = 0)
(rq = 1) (rr = 4) (st = 0) (zq = 4))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 0) (pb = 1) (re = 0)
(rq = 1) (rr = 3) (st = 0) (zq = 4))
pred: ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 4) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 3))
pred: ((bi = 2) (br = 4) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 1) (pb = 1) (re = 0)
(rq = 1) (rr = 2) (st = 0) (zq = 3))

Class: (p)
number of nodes observed: 13

pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = (1 2)) (dr = 4) (np = 1) (pb = (0 1)) (re = 0)
(rq = (1 3)) (rr = (2 4)) (st = 0) (zq = (3 4)))
evidence: 5 conflict: 0
opposing observations:

observations not included:
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 3) (np = 1) (pb = 1) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 2))
pred: ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 4) (dr = 4) (np = 1) (pb = 0) (re = 0)
(rq = 3) (rr = 2) (st = 1) (zq = 1))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 3) (np = 1) (pb = 1) (re = 0)
(rq = 3) (rr = 2) (st = 0) (zq = 3))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 3) (np = 1) (pb = 1) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 2))
pred: ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 1) (pb = 1) (re = 0)
(rq = 2) (rr = 3) (st = 0) (zq = 4))
pred: ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 3) (dr = 4) (np = 0) (pb = 0) (re = 0)
(rq = 3) (rr = 2) (st = 1) (zq = 1))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 3) (np = 1) (pb = 1) (re = 0)
(rq = 2) (rr = 3) (st = 0) (zq = 4))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 4) (dr = 3) (np = 1) (pb = 1) (re = 0)
(rq = 3) (rr = 2) (st = 0) (zq = 3))

Class: (b)
number of nodes observed: 38

pred: ((bi = (2 4)) (br = (2 5)) (bu = 0) (bz = (0 1))
(dl = (1 2)) (dr = (3 4)) (np = (0 1)) (pb = (0 1)) (re = 0)
(rq = (1 3)) (rr = (2 3)) (st = (0 1)) (zq = (2 4)))
evidence: 26 conflict: 9
opposing observations:
(d)--> ((bi = 2) (br = 4) (bu = 0) (bz = 1)
(dl = 1) (dr = 3) (np = 0) (pb = 1) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 3))
(v)--> ((bi = 2) (br = 3) (bu = 0) (bz = 1)
(dl = 2) (dr = 4) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 3) (st = 1) (zq = 3))
(d)--> ((bi = 2) (br = 3) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)

(rq = 3) (rr = 2) (st = 1) (zq = 3))
(d)--> ((bi = 4) (br = 5) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 4))
(p)--> ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 1) (pb = 1) (re = 0)
(rq = 2) (rr = 3) (st = 0) (zq = 4))
(d)--> ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 0) (pb = 1) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 3))
(v)--> ((bi = 2) (br = 4) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 1) (pb = 1) (re = 0)
(rq = 1) (rr = 2) (st = 0) (zq = 3))
(d)--> ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 3))
(d)--> ((bi = 2) (br = 5) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 3))

observations not included:
pred: ((bi = 4) (br = 5) (bu = 0) (bz = 1)
(dl = 2) (dr = 4) (np = 1) (pb = 0) (re = 1)
(rq = 1) (rr = 2) (st = 0) (zq = 4))
pred: ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 2) (dr = 4) (np = 0) (pb = 0) (re = 1)
(rq = 1) (rr = 2) (st = 0) (zq = 4))
pred: ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 1) (pb = 1) (re = 0)
(rq = 1) (rr = 4) (st = 0) (zq = 4))
pred: ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 2) (dr = 1) (np = 1) (pb = 1) (re = 0)
(rq = 4) (rr = 2) (st = 0) (zq = 1))
pred: ((bi = 2) (br = 5) (bu = 0) (bz = 0)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 1)
(rq = 2) (rr = 2) (st = 1) (zq = 3))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 3) (np = 1) (pb = 1) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 4))
pred: ((bi = 1) (br = 3) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = 1) (zq = 1))
pred: ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 1) (pb = 0) (re = 0)
(rq = 1) (rr = 4) (st = 0) (zq = 3))
pred: ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = 0) (zq = 1))
pred: ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 1) (dr = 3) (np = 1) (pb = 1) (re = 1)

(rq = 1) (rr = 2) (st = 0) (zq = 4))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 4) (np = 1) (pb = 1) (re = 0)
(rq = 1) (rr = 4) (st = 0) (zq = 5))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 4) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 4))

Class: (d)
number of nodes observed: 21

pred: ((bi = (2 3)) (br = (3 5)) (bu = (0 1)) (bz = 1)
(dl = (2 4)) (dr = (2 3)) (np = (0 1)) (pb = (0 1)) (re = 0)
(rq = (2 4)) (rr = (1 2)) (st = (0 1)) (zq = (1 3)))
evidence: 11 conflict: 4
opposing observations:
(b)--> ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = 0) (zq = 1))
(v)--> ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 4) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 3))
(b)--> ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 3))
(b)--> ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 3))

pred: ((bi = (2 3)) (br = (3 5)) (bu = (0 1)) (bz = 1)
(dl = (1 4)) (dr = (2 3)) (np = (0 1)) (pb = (0 1)) (re = 0)
(rq = (2 4)) (rr = (1 2)) (st = (0 1)) (zq = (2 3)))
evidence: 9 conflict: 4
opposing observations:
(b)--> ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 1) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 3) (rr = 2) (st = 0) (zq = 3))
(v)--> ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 4) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 3))
(b)--> ((bi = 3) (br = 5) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 3))
(b)--> ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 3))

observations not included:

pred: ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 1) (pb = 0) (re = 0)
(rq = 1) (rr = 4) (st = 0) (zq = 4))
pred: ((bi = 3) (br = 5) (bu = 0) (bz = 0)
(dl = 1) (dr = 4) (np = 1) (pb = 0) (re = 1)
(rq = 1) (rr = 3) (st = 1) (zq = 4))
pred: ((bi = 2) (br = 5) (bu = 0) (bz = 0)
(dl = 2) (dr = 1) (np = 1) (pb = 1) (re = 0)
(rq = 4) (rr = 1) (st = 0) (zq = 1))
pred: ((bi = 4) (br = 5) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 1) (pb = 1) (re = 1)
(rq = 1) (rr = 2) (st = 0) (zq = 4))
pred: ((bi = 4) (br = 5) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 4))
pred: ((bi = 3) (br = 5) (bu = 1) (bz = 1)
(dl = 2) (dr = 4) (np = 0) (pb = 0) (re = 0)
(rq = 1) (rr = 3) (st = 1) (zq = 3))
pred: ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 3) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = 1) (zq = 1))
pred: ((bi = 1) (br = 1) (bu = 1) (bz = 0)
(dl = 2) (dr = 3) (np = 0) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = 1) (zq = 1))
pred: ((bi = 3) (br = 4) (bu = 0) (bz = 1)
(dl = 1) (dr = 4) (np = 0) (pb = 1) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 3))

Class: (e)
number of nodes observed: 29

pred: ((bi = (1 2)) (br = (2 3)) (bu = (0 1)) (bz = 0)
(dl = (1 4)) (dr = (2 4)) (np = (0 2)) (pb = 0) (re = 0)
(rq = (1 4)) (rr = (2 3)) (st = (0 2)) (zq = (1 4)))
evidence: 23 conflict: 9
opposing observations:
(p)--> ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 4) (dr = 4) (np = 1) (pb = 0) (re = 0)
(rq = 3) (rr = 2) (st = 1) (zq = 1))
(k)--> ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 1) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = 1) (zq = 1))
(d)--> ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 3) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = 1) (zq = 1))
(p)--> ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 4) (np = 1) (pb = 0) (re = 0)
(rq = 1) (rr = 2) (st = 0) (zq = 3))

(p)--> ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 3) (dr = 4) (np = 0) (pb = 0) (re = 0)
(rq = 3) (rr = 2) (st = 1) (zq = 1))
(b)--> ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 4) (np = 0) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 1) (zq = 4))
(p)--> ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 4) (np = 1) (pb = 0) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 3))
(t)--> ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 2) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 2) (rr = 3) (st = 1) (zq = 1))
(k)--> ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = 0) (zq = 1))

observations not included:
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 4) (np = 0) (pb = 1) (re = 0)
(rq = 1) (rr = 3) (st = 0) (zq = 3))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 4) (np = 0) (pb = 1) (re = 0)
(rq = 3) (rr = 2) (st = 0) (zq = 2))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 1) (dr = 4) (np = 0) (pb = 1) (re = 0)
(rq = 1) (rr = 3) (st = 0) (zq = 4))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 4) (np = 1) (pb = 1) (re = 0)
(rq = 3) (rr = 2) (st = 0) (zq = 2))
pred: ((bi = 1) (br = 2) (bu = 1) (bz = 1)
(dl = 1) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 1) (rr = 3) (st = 2) (zq = 4))
pred: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 4) (np = 0) (pb = 1) (re = 0)
(rq = 2) (rr = 2) (st = 0) (zq = 1))

Confusion Areas

classes: ((b) (e))
confusion: ((bi = 2) (br = (2 3)) (bu = 0) (bz = 0)
(dl = (1 2)) (dr = (3 4)) (np = (0 1)) (pb = 0) (re = 0)
(rq = (1 3)) (rr = (2 3)) (st = (0 1)) (zq = (2 4)))
number of observations within:

classes: ((e) (p))
confusion: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = (1 2)) (dr = 4) (np = 1) (pb = 0) (re = 0)
(rq = (1 3)) (rr = (2 3)) (st = 0) (zq = (3 4)))

number of observations within: (p)=2

classes: ((e) (v))
confusion: ((bi = (1 2)) (br = 3) (bu = (0 1)) (bz = 0)
(dl = (1 2)) (dr = (3 4)) (np = (0 1)) (pb = 0) (re = 0)
(rq = (1 3)) (rr = 3) (st = (1 2)) (zq = (1 4)))
number of observations within:

classes: ((e) (t))
confusion: ((bi = 1) (br = 2) (bu = 1) (bz = 0)
(dl = 2) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 2) (rr = 3) (st = 1) (zq = 1))
number of observations within: (t)=1

classes: ((e) (k))
confusion: ((bi = 1) (br = 2) (bu = (0 1)) (bz = 0)
(dl = (1 2)) (dr = 3) (np = (1 2)) (pb = 0) (re = 0)
(rq = 4) (rr = 2) (st = (0 1)) (zq = 1))
number of observations within: (k)=2

classes: ((e) (k))
confusion: ((bi = 1) (br = 2) (bu = (0 1)) (bz = 0)
(dl = (1 4)) (dr = (2 3)) (np = 1) (pb = 0) (re = 0)
(rq = (3 4)) (rr = (2 3)) (st = (0 1)) (zq = 1))
number of observations within: (k)=2 (d)=1

classes: ((b) (d))
confusion: ((bi = (2 3)) (br = (3 5)) (bu = 0) (bz = 1)
(dl = 2) (dr = 3) (np = (0 1)) (pb = (0 1)) (re = 0)
(rq = (2 3)) (rr = 2) (st = (0 1)) (zq = (2 3)))
number of observations within: (b)=2 (d)=3

classes: ((d) (g))
confusion: ((bi = 3) (br = 3) (bu = 1) (bz = 1)
(dl = 2) (dr = 3) (np = (0 1)) (pb = 0) (re = 0)
(rq = (2 4)) (rr = 2) (st = 1) (zq = (1 2)))
number of observations within:

classes: ((b) (d))
confusion: ((bi = (2 3)) (br = (3 5)) (bu = 0) (bz = 1)
(dl = (1 2)) (dr = 3) (np = (0 1)) (pb = (0 1)) (re = 0)
(rq = (2 3)) (rr = 2) (st = (0 1)) (zq = (2 3)))
number of observations within: (b)=3 (d)=4

classes: ((d) (g))
confusion: ((bi = 3) (br = 3) (bu = 1) (bz = 1)
(dl = (1 2)) (dr = 3) (np = (0 1)) (pb = 0) (re = 0)
(rq = (2 4)) (rr = 2) (st = 1) (zq = 2))
number of observations within:

classes: ((b) (v))
confusion: ((bi = (2 4)) (br = (3 4)) (bu = 0) (bz = (0 1))
(dl = (1 2)) (dr = (3 4)) (np = (0 1)) (pb = 0) (re = 0)
(rq = (1 3)) (rr = 3) (st = 1) (zq = (2 4)))
number of observations within: (v)=1 (b)=1

classes: ((k) (t))
confusion: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 3) (np = 1) (pb = 1) (re = 0)
(rq = 4) (rr = 1) (st = 0) (zq = 1))
number of observations within:

classes: ((3) (k))
confusion: ((bi = 1) (br = 2) (bu = 0) (bz = 0)
(dl = 2) (dr = 3) (np = 1) (pb = 0) (re = 0)
(rq = 4) (rr = 1) (st = 0) (zq = 1))
number of observations within: (3)=1

classes: ((b) (e) (v))
confusion: ((bi = 2) (br = 3) (bu = 0) (bz = 0)
(dl = (1 2)) (dr = (3 4)) (np = (0 1)) (pb = 0) (re = 0)
(rq = (1 3)) (rr = 3) (st = 1) (zq = (2 4)))
number of observations within:

Appendix B.

Other Results

The following is an excerpt of tests run on the learning system to illustrate its learning capacity when faced with erroneous data. Three regions of the plane (A, B, and C) were delineated with overlaps covering 13% of their total area. 100 points were randomly generated within the space, and 6 randomly chosen points were deliberately mislabelled to create errors. These points were used as learning data.

Generalizations were produced (including confusion areas) in seperate runs involving 2 permutations of the learning examples, and with 4 different error tolerances. The generalizations produced are here called filters to reflect their role in classifying a second set of randomly generated points within the space.

The second set of points were compared to the learned generalizations and were classified according to the encompassing generalization or in cases where they did not fall into a generalization (or confusion area) according to the nearest generalization. The classification according to the learned generalizations was compared to the real membership of the points. This produced correct classifications, near-correct classifications (when the learned generalization was nearest to the point), and similarly confused, near-confused, incorrect and near-incorrect classifications.

The first two tables show the ideal results of the learning data and the testing data when filtered through the true characterizations of the regions. They provide a record on the accuracy of the data and of the best that might reasonably be expected of learned filters. Among the 4 error tolerances used is one that allowed any and all alien points intruding into a generalization. The results in this case were the same for both permutations of the data. This provides a view of the classification that results from complete generalization.

The results show that the system is capable of producing useful generalization even when errors exist. The most interesting result is that learning seemed to be at its best when the error tolerance chosen was closest to the real error.

learn data true filter results

	overall	A	B	C
tests:	100	34	33	33
correct:	64	22	22	20
confused:	30	10	9	11
incorrect:	6	2	2	2

test data true filter results

	overall	A	B	C
tests:	50	16	17	17
correct:	37	14	10	13
confused:	13	2	7	4
incorrect:	0	0	0	0

data perm 1 tolerance a (< 10 abs., < 1/2 rel.)
learned filter

	overall	B	C	A
tests:	50	17	17	16
correct:	42	12	16	14
near-correct:	0	0	0	0
confused:	4	3	1	0
near-confused:	0	0	0	0
incorrect:	4	2	0	2
near-incorrect:	0	0	0	0
unclassified:	0	0	0	0

data perm 1 tolerance b (< 5 abs., < 1/3 rel.)
learned filter

	overall	B	C	A
tests:	50	17	17	16
correct:	37	12	11	14

near-correct:	6	3	3	0
confused:	0	0	0	0
near-confused:	1	0	0	1
incorrect:	1	0	0	1
near-incorrect:	5	2	3	0
unclassified:	0	0	0	0

data perm 1 tolerance d (complete, anything)
learned filter

	overall	B	C	A
tests:	50	17	17	16
correct:	6	2	1	3
near-correct:	0	0	0	0
confused:	44	15	16	13
near-confused:	0	0	0	0
incorrect:	0	0	0	0
near-incorrect:	0	0	0	0
unclassified:	0	0	0	0

data perm 2 tolerance a (< 10 abs., < 1/2 rel.)
learned filter

	overall	B	A	C
tests:	50	17	16	17
correct:	29	12	4	13
near-correct:	10	0	7	3
confused:	4	3	0	1
near-confused:	0	0	0	0
incorrect:	4	2	2	0
near-incorrect:	3	0	3	0
unclassified:	0	0	0	0

data perm 2 tolerance b (< 5 abs., < 1/3 rel.)
learned filter

	overall	B	A	C
tests:	50	17	16	17
correct:	25	12	2	11
near-correct:	14	0	9	5
confused:	0	0	0	0
near-confused:	1	0	0	1
incorrect:	5	5	0	0
near-incorrect:	5	0	5	0

unclassified:	0	0	0	0

data perm 2 tolerance c (< 2 abs., < 1/10 rel.)
learned filter

	overall	B	A	C
tests:	50	17	16	17
correct:	22	9	7	6
near-correct:	20	3	7	10
confused:	0	0	0	0
near-confused:	0	0	0	0
incorrect:	4	4	0	0
near-incorrect:	4	1	2	1
unclassified:	0	0	0	0