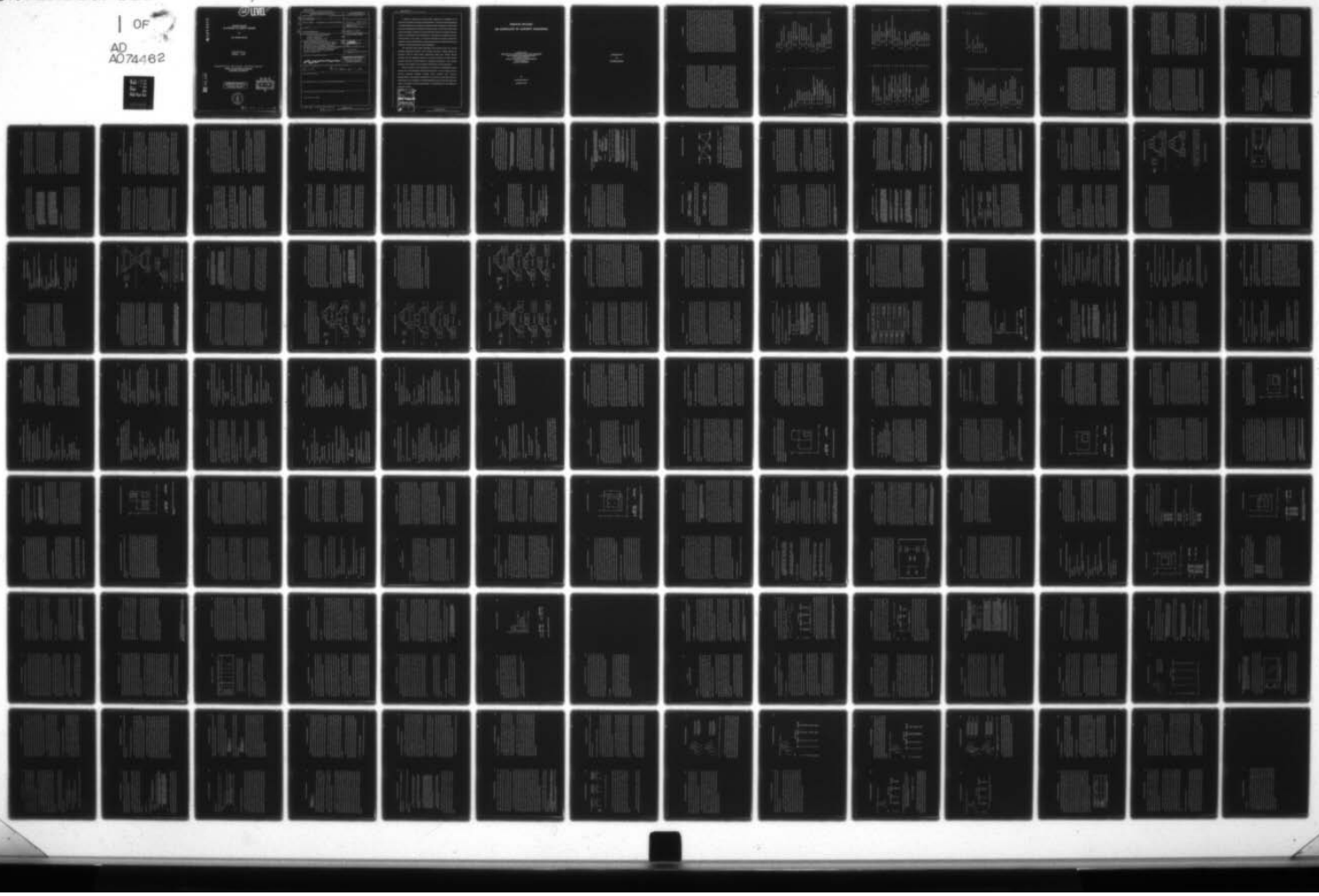
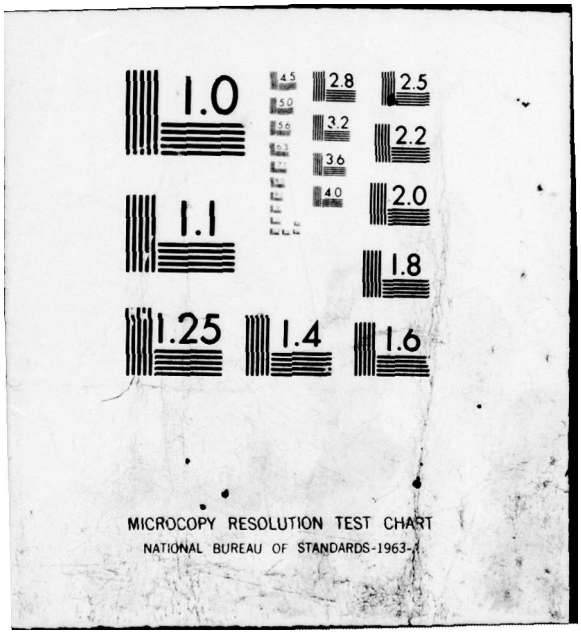
AD-A074 462 STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE F/G 9/4
VERSION SPACES: AN APPROACH TO CONCEPT LEARNING.(U)
DEC 78 T M MITCHELL DAHC15-73-C-0435
UNCLASSIFIED STAN-CS-78-711 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

VERSION SPACES: AN APPROACH TO CONCEPT LEARNING

by

Tom Michael Mitchell

STAN-CS-78-711
December 1978

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) STAN-CS-78-711	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Version Spaces: An Approach to Concept Learning		5. TYPE OF REPORT & PERIOD COVERED technical, December 1978
		6. PERFORMING ORG. REPORT NUMBER STAN-CS-78-711
7. AUTHOR(s) (10) Tom Michael Mitchell		8. CONTRACT OR GRANT NUMBER(s) (15) DAHC15-73-C-0435
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department Stanford University Stanford, California 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 094 120
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Information Processing Techniques Office 1400 Wilson Ave., Arlington, VA 22209		(11) 12. REPORT DATE Dec[redacted]78 13. NUMBER OF PAGES 216
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Mr. Philip Surra, Resident Representative Office of Naval Research, Durand 165 Stanford University		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Reproduction in whole or in part is permitted for U.S. Government purposes.

DISTRIBUTION STATEMENT A
Approved for public release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(12) 212 (9) Technical repts

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(see reverse side)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

094120

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

A method is presented for learning general descriptions of concepts from a sequence of positive and negative training instances. This method involves examining a predetermined space or language of possible concept descriptions, finding those which are consistent with the observed training instances. Rather than use heuristic search techniques to examine this concept description space, the subspace (version space) of all plausible concept descriptions is represented and updated with each training instance. This version space approach determines all concept descriptions consistent with the training instances, without backtracking to reexamine past training instances or previously rejected concept descriptions.

The computed version space summarizes the information within the training instances concerning the identity of the concept to be learned. Version spaces are therefore useful for making reliable classifications based upon partially learned concepts, and for proposing informative new training instances to direct further learning. The uses of *version* spaces for detecting inconsistency in the training instances, and for learning in the presence of inconsistency are also described.

Proofs are given for the correctness of the method for representing version spaces, and of the associated concept learning algorithm, for any countably infinite concept description language. Empirical results obtained from computer implementations in two domains are presented. The version space approach has been implemented as one component of the Meta-DENDRAL program for learning production rules in the domain of chemical spectroscopy. Its implementation in this program is described in detail.

VERSION SPACES:

AN APPROACH TO CONCEPT LEARNING

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By

Tom Michael Mitchell

December, 1978

Abstract

A method is presented for learning general descriptions of concepts from a sequence of positive and negative training instances. This method involves examining a predetermined space or language of possible concept descriptions, finding those which are consistent with the observed training instances. Rather than use heuristic search techniques to examine this concept description space, the subspace (version space) of all plausible concept descriptions is represented and updated with each training instance. This version space approach determines all concept descriptions consistent with the training instances, without backtracking to reexamine past training instances or previously rejected concept descriptions.

The computed version space summarizes the information within the training instances concerning the identity of the concept to be learned. Version spaces are therefore useful for making reliable classifications based upon partially learned concepts, and for proposing informative new training instances to direct further learning. The uses of version spaces for detecting inconsistency in the training instances, and for learning in the presence of inconsistency are also described.

Proofs are given for the correctness of the method for representing version spaces, and of the associated concept learning algorithm, for any countably infinite concept description language. Empirical results obtained from computer implementations in two domains are presented. The version space approach has been implemented as one component of the Meta-DENDRAL program for learning production rules in the domain of chemical spectroscopy. Its implementation in this program is described in detail.

Acknowledgements

It is impossible to find space to thank all the people who have contributed toward this work. Some of those who I especially want to thank are: Bruce Buchanan. This man is a philosopher, a scientist, and a friend. Three years of his insights, guidance, and moral support allowed this work to occur. Cordell Green, Richard Pantell, and Edward Feigenbaum, the other members of my reading and orals committees, whose comments aided in clarifying the presentation of the ideas in this thesis. Reid Smith and Jan Aikins for their support of every imaginable kind. Their help in figuring out what I was doing, their encouragement that it was worth doing, and their friendship helped put things in perspective when it was needed most. Others who took the time to read and comment carefully on various drafts of this dissertation: Saul Amarel, John Burge, Lew Creary, Jim Davidson, Peter Friedland, Ann Gardner, Rick Hayes-Roth, Chitoor Srinivasan, and Mark Stefik. My parents, who have helped me in countless ways, and whose enthusiasm for learning has been a great influence. Joan, my best friend and wife. She kept us human during all the late nights and crazy weeks. I'll never be sure whether this thesis took more of my time or hers. Thanks as well to SUMEX-AIM, the Stanford Artificial Intelligence Laboratory, and Rutgers University for providing facilities for generating this text. Special thanks to Reid Smith who supplied the PUB commands for formatting this document, and to Lew Creary who kindly ran off the final copy. Various phases of this work were supported by the National Institutes of Health under grant RR 00612-07 and by the Advanced Research Projects Agency under contract DAHC 15-73-C-0435.

Table of Contents

Chapter 1

Introduction

Machine learning is one of the earliest and most ambitious goals of computer science. In the past two decades, several partially successful programs have been written that are capable of various kinds of learning in well-controlled, constrained domains (e.g., [Samuel, 1963], [Feigenbaum, 1963], [Hunt, 1975], [Plotkin, 1970], [Winston, 1970], [Waterman, 1970], [Michalski, 1973], [Hayes-Roth, 1974], [Vere, 1975], [Soloway, 1977], [Buchanan, 1974], [Lenat, 1976], [Langley, 1977]). The results so far have been tantalizing. The plausibility of machine learning in narrowly restricted domains has been demonstrated, but the proficiency and generality of the learning strategies which have been employed have yet to be determined. Although progress has been made toward the goal of machine learning, we are only beginning to understand many of the problems involved and to discover methods for dealing with these problems.

One recent trend in artificial intelligence (AI) research provides strong practical motivation for research in machine learning. The increasing success of performance programs that make extensive use of domain-specific knowledge underscores the importance of developing efficient methods for *acquiring* such knowledge. One performance program which employs domain-specific knowledge is the DENDRAL program [Feigenbaum, 1971], which aids chemists in determining the molecular structure of unknown compounds. This program utilizes its knowledge of chemistry to infer plausible molecular structures consistent with available laboratory data

associated with an unknown sample. The task of acquiring the domain specific knowledge used by DENDRAL and other "knowledge-based" programs (e.g., MYCIN [Shortliffe, 1976], HEARSAY [Lesser, 1976]) has proven a difficult and time consuming task. Informal methods for obtaining domain-specific knowledge by querying experts (e.g., chemists, doctors) often result in inexact, incomplete, and inconsistent knowledge bases [Feigenbaum, 1977]. One recent study [Davis, 1976] has focussed on methods for systematizing and automating the querying of experts to obtain reliable knowledge bases.

Learning programs offer an attractive alternative to obtaining domain knowledge solely from human experts. A program capable of modifying existing rules and inferring new rules for inclusion in the knowledge bases of programs such as DENDRAL would be an important asset for increasing the proficiency and flexibility of these programs.

Building such learning programs is a well-defined and realistic goal for research on machine learning - and a goal toward which some progress has already been made. The Meta-DENDRAL program [Buchanan, 1978], for instance, learns production rules that contain one kind of chemical knowledge used by the DENDRAL program. The rules that Meta-DENDRAL learns are used to predict peaks in the mass spectra and ^{13}C NMR spectra of classes of organic compounds. Programs that acquire domain-specific knowledge for performance programs in other areas have also been written (e.g., playing draw poker [Waterman, 1970], diagnosing plant diseases [Larson, 1977], recognizing spoken words [Hayes-Roth, 1976]).

1.1 The Problem Considered Here

"Learning" is a broad term covering a wide range of processes. We learn (memorize) multiplication tables, learn (discover how) to walk, learn (build up an understanding of, then an ability to synthesize) language. Our current understanding of learning is sketchy, and it would be premature to try to program a computer to attempt very sophisticated learning problems without first developing a better understanding of the central processes involved in learning. This study therefore focuses on one type of behavior that is central to a wide range of learning tasks: the ability to ***generalize***.

The ability to generalize is the ability to take into account many specific experiences and observations, then to extract and retain the important common features of those experiences. For instance, a child learns to associate classes of objects or actions with words. He sees many kinds of birds (and many more things which are not birds) and forms a general notion or concept of "bird" that he learns to associate with the correct word, and which allows him to identify birds he has never seen. A research chemist performs a similar task in attempting to characterize the behavior of molecules in analytic instruments. He begins by examining the structure and exhibited analytic data of many known molecules, then extracts general notions or rules which allow him to predict the behavior (and observable data) for molecules which he has never examined.

This ability to generalize from specific instances of a class to a general model or description of the class, often referred to as ***concept learning***, is the central topic of

this dissertation. Concept learning has been studied by many researchers in the fields of psychology (e.g., [Bruner, 1956], [Hunt, 1975]) and artificial intelligence (e.g., [Evans, 1968],[Waterman, 1970],[Winston, 1975],[Michalski, 1973],[Hayes-Roth, 1975], [Vere, 1977]). The ability to generalize from examples is both a powerful ability in itself, and a key building block for more sophisticated learning behavior. For example, the generalization techniques developed in this dissertation have been used as a central component of the Meta-DENDRAL program for inferring production rules to be used by DENDRAL.

1.1.1 Representing Concepts

The ability to learn concepts presupposes the ability to represent and store learned concepts. In this work, concepts are described by patterns which state the properties common to instances of the concept. The utility of representing concepts in this way has been demonstrated by earlier concept learning programs in a variety domains (e.g., [Winston, 1970],[Michalski, 1973],[Hayes-Roth, 1975],[Vere, 1975], [Buchanan, 1978]). In this work, as with earlier work, the primitive properties and relations appropriate for describing concepts in any given domain are supplied to the program, and define a language in which the program must describe concepts. This language is referred to here as the ***concept description language***, or equivalently, the ***pattern language***.

1.1.2 Concept Learning

In order to learn a particular concept, a program must have both a language in

which to express concepts and a set of training instances that exemplify some "target concept". Training instances are labeled as either positive instances (examples of the concept) or negative instances (not examples of the concept). The task of the concept learning program is to determine a description of the target concept that is consistent with these observed training instances. In summary, the class of concept learning problems considered here is the following:

Concept Learning Problem:

Given: 1. A concept description language.

2. A procedure that matches concept descriptions to training instances.

3. Sets of positive and negative training instances of the target concept.

Determine: Concept descriptions within the given language which are consistent with the provided training instances.

Concept learning as defined above can be viewed as a search problem. The language of allowed concept descriptions defines the domain of concepts that the program might learn, or the search space of possible "solutions" to the concept learning problem. The program must examine this solution space, subject to constraints imposed by the training instances, to determine a valid description of the target concept. Many researchers have shared the view of concept learning as a search problem [Amarel, 1971], [Winston, 1970], [Simon, 1973], [Hayes-Roth, 1975].

1.2 Some Earlier Approaches

If concept learning is viewed as a search problem, then concept learning methods can be characterized in terms of the search strategies which they employ. The two most common classes of search strategies employed by concept learning programs described in the literature can be characterized as depth-first search, and breadth-first search. General capabilities and limitations associated with each of these classes of strategies are described in chapter 7. The major results of chapter 7 are summarized here to provide an indication of the capabilities and limitations of existing approaches to concept learning. References to several programs are included in the discussion of each class of search strategies. Although none of these programs employs exactly the same strategy, there are programs that share capabilities and limitations characteristic of definite classes.

1.2.1 Depth-First Search

One common strategy for examining the space of possible concept descriptions is *depth-first search* (see, for instance, [Winston, 1970], [Waterman, 1970]), in which a single concept description from the search space is chosen as the *current best hypothesis* to describe the target concept. This concept description is then tested against each of a sequence of training instances. If the concept description is inconsistent with a new training instance, it is modified to become consistent with the new training instance while remaining consistent with past training instances. Each such modification yields a new current best hypothesis, and can be viewed as a step in a depth-first search through the search space of possible concept descriptions.

Depth-first search was one of the earliest approaches to learning concepts described as patterns [Winston, 1970]. Although this method is useful, it has several shortcomings. The following two difficulties are inherent to the depth-first search strategy for concept learning.

1) In modifying the current concept description in response to a new training instance, it is costly to determine which possible modifications are consistent with previous training instances. Each modification must be tested against all previous training instances. As a result, the more training instances the program has processed, and therefore the closer it is to learning the correct description of the concept, the greater the cost of processing the next training instance.

2) Once the program has determined which modifications will result in concept descriptions consistent with the training instances, it must choose one of these concept descriptions as the new current best hypothesis. Since it is impossible to reliably choose the "correct" modification from among the many possibilities, the program must be prepared to backtrack and reconsider alternate choices if subsequent training instances reveal that an incorrect choice has been made.

1.2.2 Breadth-First Search

Several researchers [Plotkin, 1970], [Michalski, 1973], [Hayes-Roth, 1974], [Vere, 1975] have written concept learning programs employing a *breadth-first search* strategy for examining the space of possible concept descriptions. Each of these programs takes advantage of an important structure on the search space of possible concept descriptions to organize the breadth-first search. This structure is a partial ordering of concept descriptions which embodies the usual intuitive notion of the terms "more general" and "more specific". These programs begin with the most

specific concept description consistent with the first training instance, then consider progressively more general concept descriptions, as needed, to determine the set of maximally specific concept descriptions consistent with the training instances. Thus, the program sweeps out a breadth-first search from specific to general concept descriptions.

Because the breadth-first search is organized around the specific-to-general partial ordering on the search space, modifications to the set of current hypotheses can be made without reconsidering past *positive* training instances. Past negative instances must, however, still be reexamined. The breadth-first search programs cited above represent an important step in taking advantage of the specific-to-general partial ordering inherent to this class of concept learning problems: (1) to limit the number of considered modifications to the set of current hypotheses, and (2) to avoid the need to reconsider past *positive* training instances when processing new training instances.

1.3 Outstanding Problems

Despite the progress which has been made toward finding good strategies for concept learning, many critical problems remain. In addition to the central problem of searching an enormous space of possible concept descriptions for one consistent with the training instances, there are more global problems. These include determining the point at which a concept has been reliably learned, detecting and recovering from inconsistent training data, and proposing informative new training instances. Solutions to these problems require that the program be capable of examining what it has

learned so far - of summarizing and reasoning with the *partial* information that has been conveyed by the training instances concerning the identity of the target concept. Each of these problems, restated as a question, is considered below. Progress toward solutions to these problems forms the main focus of this thesis.

1.3.1 Which Concept Descriptions Are Consistent With the Training Instances?

The central problem faced by any concept learning program is to determine a general description of the target concept which is consistent with the observed training instances. Additional criteria for acceptability such as "simplicity", "elegance" or "generality" may also be important in choosing a concept description, but the primary criterion is consistency with the training instances.

For both the depth-first search and breadth-first search strategies described above, an important and costly step is determining *which modifications to a current hypothesis are consistent with past training instances*. For the depth-first search programs referenced, each past training instance must be reexamined to obtain this information. With the above breadth-first search programs, each past *negative* instance must be reexamined. Is there a compact method of summarizing the information contained in the training instances so that past instances do not have to be reexamined at all? Is there a way of efficiently keeping track of *all* concept descriptions consistent with past training instances?

The term ***version space*** is used in this work to refer to the set of all concept descriptions, within a prescribed language, that are consistent with a set of training

instances. The following chapters describe and prove correct a method for efficiently representing and computing version spaces.

1.3.2 When is the Concept Unambiguously Learned?

"How can a concept learning program detect the point at which it has observed enough training instances to reliably use the learned concept?" For programs which employ a depth-first search approach, the question becomes "How can the program predict whether additional training instances will lead to further modifications to the current hypothesis?". This is a crucial question for gauging the reliability of classifications based upon the learned concept description.

To answer this question the program must be able to detect whether there are concept descriptions in addition to the current hypothesis that are consistent with the training instances. If so, then the identity of the concept has only been partially determined by the training instances, and the concept is not yet completely learned. If additional training instances are not available, then a second question arises: "How can descriptions of partially learned concepts be used to classify future instances in a reliable manner?" For some concept description languages a finite set of training instances can never determine a unique concept description. For such problems, the issue of reliable use of partially learned concept descriptions is unavoidable.

Chapter 4 illustrates the use of version spaces to detect whether a unique concept description is determined by the observed instances. The use of version spaces to determine classifications of new instances based upon partially learned concepts is also described.

1.3.3 Are the Training Instances Consistent?

In many problem domains, learning is complicated by the inconsistency of the training instances with respect to the language of concept descriptions. If the training instances contain errors, or the language of concept descriptions is insufficient, then there may be *no* concept descriptions consistent with the training instances. *In such inconsistent concept learning problems the program must be able* to detect and recover from inconsistency.

How can a concept learning program efficiently detect this kind of inconsistency? What does it mean to learn a concept in the presence of inconsistency? Is there an efficient method for finding concept descriptions that are "maximally consistent" with the training instances?

Chapters 3 and 5 describe and prove correct a method for determining concept descriptions that are maximally consistent with a set of training instances. A more efficient, approximate method, based upon this exact method, is also presented.

1.3.4 What Additional Training Instances Would be Informative?

Many authors (e.g., [Simon, 1973],[Smith, 1977], [Buchanan, 1974],[Winston, 1970]) have stressed the significance of a carefully chosen sequence of training instances to good learning behavior. Winston, for example, discusses the importance of showing the program instances which are "near misses" of the target concept; that is, negative training instances which differ in only small ways from positive instances of the concept.

An important question is whether the program itself can propose instances whose classification as positive or negative instances will reveal additional information concerning the identity of the target concept. The strategy which a program must employ to chose its own training instances is much different from that which a good teacher must employ - the program must choose informative training instances *without* knowing the identity of the target concept (see the discussion of learning with and without a teacher in [Buchanan, 1978a]). To choose informative new training instances, the program must be aware of *what cannot be determined on the basis of observed data* concerning identity of the target concept.

Chapter 4 describes a strategy for proposing optimally informative new training instances, based upon examining the version space determined by past observed instances.

1.4 Version Spaces and Concept Learning

Answers for each of the above four questions are possible if the learning program represents explicitly *what can and cannot be determined* about the identity of the target concept. Version spaces summarize this needed information.

The approach to concept learning proposed here involves representing and updating the version space of all concept descriptions consistent with the observed training instances. The version space is initialized to contain all concept descriptions consistent with the first positive training instance. Additional training instances are then considered one at a time. Candidate concept descriptions are eliminated from the

version space as they are found to be inconsistent with subsequent training instances, so that at each step, exactly those concept descriptions consistent with the training instances remain.

Since the version space may in general contain many concept descriptions, an efficient method for representing version spaces is needed. A general (independent of the concept description language) scheme for representing version spaces and for revising a represented version space in response to subsequent training instances is presented and proven correct.

1.5 Main Themes of This Work

In considering issues related to concept learning, many intermingled themes appear in the following chapters. Several of these themes are described below.

Concept learning as search. Concept learning is a problem of examining a prescribed space of possible solutions (concept descriptions) subject to a set of constraints on acceptable solutions (training instances). This view of concept learning provides a useful perspective for comparing alternate approaches to concept learning (see chapter 7), as well as for relating concept learning to other problem solving tasks (see [Simon, 1973]).

Version space approach as one method for examining the solution space. If concept learning is viewed as a search problem, then the version space approach to concept learning is one method for examining the solution space of concept descriptions. It is a general, theoretically sound, method which provides a coherent

way of thinking about what learning means in terms of the search paradigm. No claims are made for the psychological validity of this approach.

Version space as summary of allowed concept descriptions. The summary of allowed concept descriptions provided by the represented version space allows the program to determine how precisely the target concept has been learned. This summary is also useful for classifying subsequent instances, even when observed training instances are insufficient to uniquely determine the concept.

Version space as summary of observed training instances. The version space provides a compact summary - in the language of concept descriptions - of the information contained in the training instances concerning the identity of the target concept. Because the version space summarizes this information, the program does not need to store observed training instances for reexamination. In addition, this view of version spaces leads to their use in detecting inconsistency in the training instances, and for combining results obtained from separate sets of training instances.

1.6 Reading the Rest

Chapter 2 introduces a general method for representing version spaces, and an associated concept learning algorithm. The method is illustrated for concept learning problems drawn from two domains.

Chapter 3 provides a formal treatment of the methods introduced in chapter 2. The correctness and generality of the version space approach to concept learning are proven. The definition of version spaces is then generalized to allow learning concepts

from inconsistent training instances, and the correctness of the associated learning algorithm (described fully in chapter 5) is proven.

Chapter 4 describes applications of version spaces to problems of reliably using partially learned concepts, proposing informative new training instances, and combining results obtained from separate sets of data.

Chapter 5 considers learning in the presence of inconsistent data. An extension to the approach introduced in chapter 2 is described, which allows determining concept descriptions maximally consistent with the observed training instances for problems involving limited inconsistency. A more efficient, approximate learning strategy based on this optimal strategy is presented for dealing with more severe inconsistency.

Chapter 6 describes in detail the implementation of the version space approach as part of the Meta-DENDRAL program for learning rules in the domain of chemical spectroscopy. Experiments are described involving the use of version spaces for concept learning in this domain.

Chapter 7 contrasts the capabilities, efficiency, and computational complexity of the version space approach with classes of earlier strategies for concept learning. The comparison is based on the view of concept learning as a search problem, and the perspective that concept learning programs may be usefully characterized in terms of their search strategies.

Chapter 8 summarizes the major contributions and limitations of the work reported here, and considers possible routes for further work.

Chapter 2

Version Spaces and Concept Learning

2.1 Introduction

This chapter considers a class of concept learning problems in which a program is presented a sequence of positive and negative training instances of the concept, and asked to construct a general description of the concept in a prescribed language. An approach to this class of problems is presented, which is assured of finding all valid descriptions of the target concept expressed in the allowed language. This version space approach to concept learning is described in general terms, and illustrated for two specific concept learning problems.

2.2 The Problem

As described in chapter 1, we consider the class of concept learning problems which may be defined as follows:

Concept Learning Problem:

Given: 1. A language of patterns for describing concepts.

2. An associated pattern matcher.

3. Sets of positive and negative training instances of the target concept.

Determine: Concept descriptions within the given language which are consistent with the training instances.

The *target concept* is the (known or unknown) concept exemplified by the set of positive training instances. We shall call the solution to the concept learning problem the *version space* of the target concept with respect to the observed training instances. That is, the version space of the target concept is the set of all plausible versions of the target concept given the observed training instances[1].

Version Space: The version space of a target concept with respect to a set of training instances is the set of all concept descriptions within the given language which are consistent with those training instances.

The term *consistent* is used above and throughout this chapter in the following strict sense: a concept description is consistent with a set of training instances if it matches *all* positive training instances, but *no* negative instances. Thus we assume that the training data contain no errors, and that the language of concept descriptions contains at least one description of the target concept consistent with all training instances. The version space approach is simplest for concept formation problems satisfying these criteria.

Chapter 5 discusses concept learning problems for which the above error-free assumption does not hold. There, an extension of the version space approach is presented in which the program attempts to find concept descriptions consistent with the largest possible subsets of the training instances.

---

[1] The term "version space" is used rather than "version set" because, as will be described, there is a useful structure on this set of concept descriptions which allows it to be efficiently represented and computed.

2.2.1 One Concept Learning Problem

In order to discuss concept learning in concrete terms, the bulk of this chapter relies upon one particular concept learning problem to illustrate general issues and techniques. This concept learning problem, originally discussed by Winston [Winston, 1970], involves learning to identify simple classes of structures built out of children's blocks. The task may be, for example, to learn the concept of an "arch" given a series of block structures, each labeled as either an arch or a non-arch.

By choosing a language for representing such "blocks world" concepts, and an associated pattern matcher, the arch learning problem can be stated as an instance of the class of concept learning problems described above. A natural choice for representing concepts involving various blocks and relationships among them is to use a language of structural patterns, or networks. Each node in the network will represent an individual block in the structure, and each link between nodes will denote some relation between the blocks. The arch learning problem can then be stated as follows:

The Arch Learning Problem:

Given: *1. Concept description language:* All possible networks with nodes representing individual blocks, and links between nodes representing relations between the corresponding blocks. Each node (block) has the following properties which may be constrained as shown:

Property	Allowed Constraints
Shape	Brick, Wedge, ..., NOT-Brick, NOT-Wedge, ..., (Brick OR Wedge), ..., any
Orientation	Standing, Lying, Leaning, ..., NOT-Standing, NOT-Lying, ..., (Standing OR Leaning), ..., any

Each link in the network is labeled as one of the following relations:

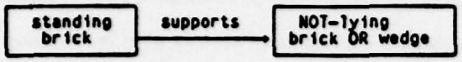
supports, does-not-support, touches, does-not-touch

2. Pattern Matcher: A pattern matches a training instance if the pattern constraints are satisfied by the instance; that is, if there is a mapping of pattern nodes onto instance nodes such that each constrained property of each node and link in the pattern is satisfied by the corresponding node or link in the instance.

3. Training instances: Network descriptions of block structures labeled as either positive or negative instances of the concept "arch".

Determine: All concept descriptions within the given language which are consistent with every training instance.

The following pattern is a legal concept description in the blocks world concept description language described above, and therefore represents a concept which the program might learn:



Training instances are represented in the same network language as the concept descriptions. For example, the following is a positive instance of the concept described above:

The relations in the concept description language are not independent. For example, the relation (A *supports* B) *implies* the relation (A *touches* B). Figure 2.1 illustrates the dependencies among these two relations and their negations. The arrows indicate which relations are implied by other relations. Because of this logical hierarchy involving the relations, only the uppermost (most specific) valid relation along each branch need be explicitly stated for any instance or concept description. All lower relations in the hierarchy are then implied by the stated relation, and all higher relations are known not to hold (or they would be the one explicitly stated).

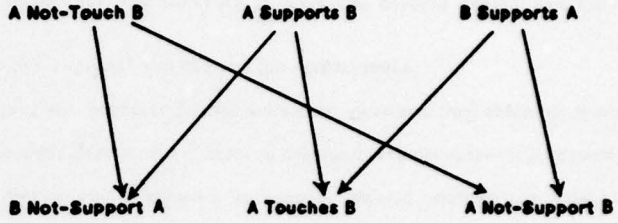


Figure 2.1 Hierarchy of "Suports" and "Touches" constraints. Arrows indicate which constraints are implied by others.

Patterns constitute sets of constraints on possible values for node and link properties. A pattern matches an instance if all the constraints stated by the pattern are satisfied by the instance. A pattern may match an instance that contains properties or relations in addition to those constrained by the pattern (i.e., the above pattern would match the above instance even if the *supports* relation was removed from the pattern). The pattern matcher must take into account the hierarchy of relations shown in figure 2.1, since this hierarchy is used for implicitly representing certain relations as described above.

2.2.1.1 A Note Concerning This Example

The details of the concept description language and the pattern matcher defined for the arch learning problem have been chosen to allow a compact example on which to focus a discussion of concept learning. Limitations of the above concept description language (i.e., the lack of additional node properties such as Size) and the pattern matcher (i.e., the requirement that pattern nodes map *onto* rather than *into* instance nodes) represent limitations on the *formulation* of this problem. These do not reflect limitations on the techniques described for concept learning, which are equally valid for more complex pattern languages and matching predicates.

2.3 The Approach

Although we have defined the arch learning problem in some detail, we have so far said little about how a program might actually learn the concept of an arch. The approach followed here is to view concept learning as a search problem[1]. The concept description language defines a search space containing descriptions of all concepts which the program might learn. A program learns the target concept by examining this search space - subject to constraints imposed by the training instances - to find valid descriptions of the target concept. Given this perspective on concept learning, there are at least as many strategies for concept learning as there are methods for examining the search space of possible concept descriptions.

The version space approach is a general method for solving the class of concept

[1] The paper [Simon, 1973] discusses learning as one form of problem solving, and search as one strategy for learning.

formation problems described above. This method accomplishes a *complete* examination of the search space of possible concept descriptions, without backtracking or reexamining training instances, assuring that all concepts consistent with the training instances will be found. In contrast with search strategies which maintain and modify a small number of concept descriptions, the version space approach efficiently keeps track of *all plausible concept descriptions* at each step. In addition to providing the focus for the learning algorithm, this computed version space contains useful information for applying partially learned concepts and for determining informative new training instances.

2.3.1 The Version Space Approach

The version space approach to concept learning reduces the initial search space of concept descriptions in order to determine exactly those descriptions which are consistent with observed training instances. Central to this approach is a means for efficiently *representing* and *updating* the version space.

The version space is initialized to contain all concept descriptions consistent with the first observed positive training instance. Subsequent positive and negative training instances are then examined, and candidate concept descriptions inconsistent with any instance are eliminated from the version space. Therefore, at each step the version space contains exactly those concept descriptions which are consistent with every observed training instance.

Features of this candidate elimination algorithm for concept learning using version spaces include:

1) All concept descriptions consistent with all training instances will be found.

2) The version space at each step provides the program with a useful summary of all alternate interpretations of the observed data. This information may be used to recognize the point at which a concept has been reliably learned, to reliably classify subsequent instances even on the basis of partially learned concepts, and to determine informative new training instances.

3) Results produced are independent of the order in which training instances are presented.

4) Each training instance is examined only once. Therefore, training instances need never be stored, and the algorithm requires time proportional to the number of observed training instances.

5) Backtracking is never required to reconsider previously examined candidate concepts.

6) The approach may be extended to handle situations in which no concept description is consistent with every training instance. Chapter 6 describes a method for determining all concept descriptions maximally consistent with the training instances. A more practical, heuristic method for dealing with inconsistency is also described.

2.3.2 Representing Version Spaces

The version space is the "solution" to the concept learning problem at any given point in the processing of training instances. Although the version space is a useful concept to work with, it can only be useful to a learning program if it can be efficiently represented and computed.

In general the version space may contain an infinite number of concept

descriptions. Even for most finite concept description languages, representing a version space by listing all its members is hopelessly inefficient. Not only will storage space requirements be prohibitive, but the processing required to check each candidate concept version against each training instance would be also be enormous. *A representation for version spaces is needed which is both compact, and easily computed by the learning algorithm.*

By a "representation" for version spaces, we mean a way of storing the information needed to reconstruct every concept description in the version space. As described below, it is possible to find such a representation for which the storage space requirements are not directly tied to the number of concept descriptions in the version space.

2.3.2.1 The General-to-Specific Partial Ordering

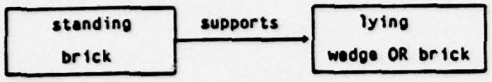
The key to finding an efficient representation for version spaces lies in observing that there is a convenient structure on the search space of concept descriptions being considered. In particular, a partial ordering can be defined for any concept description language in terms of its associated pattern matcher. The relation "more specific than or equal to" is defined as follows:

Pattern P1 is *more specific than or equal to* pattern P2 (P1 ≥ P2) if and only if P1 matches a subset of all the instances which P2 matches.

We will also use the strict relation "more specific than".

Pattern P1 is *more specific than* pattern P2 (P1 > P2) if and only if (P1 ≥ P2) and P1 is not equal to P2.

For example, in the blocks world concept language defined in the previous section, the pattern P1:



is more specific than the pattern P2:

since the constraints represented in P1 are satisfied only if the weaker constraints of P2 are satisfied. P1 matches only instances which P2 also matches, and P1 is therefore "more specific than or equal to" P2.

The above definition of "more specific" is simply a formalization of the usual intuitive notion of "*more specific*" or "less general". When these terms are used anywhere within this dissertation, they are taken to have the above well defined meaning. Notice that the definition of the partial ordering is stated in language independent terms. The "more specific than or equal to" relation is shown in chapter 3 to define a partial ordering on any concept description language. Similar partial orderings have been defined for specific pattern languages and used by [Plotkin, 1970], [Michalski, 1973], [Hayes-Roth, 1975], and [Vere, 1975] in their work involving machine learning.

2.3.2.2 The Partial Ordering for the Arch Learning Problem

It is quite easy to describe the partial ordering for the blocks-world concept description language in terms of individual constraints on nodes and links. We may define a general-to-specific ordering on allowed constraints for each node feature and on the allowed relations between nodes, based upon the meaning of each constraint as defined by the pattern matcher. Pattern P1 is then more specific than P2 if and only if each constraint in P1 is more specific than the corresponding constraint in P2.

For instance, the hierarchy shown in figure 2.1 corresponds to the general-to-specific ordering for *relations* between nodes[1]. In that figure, the arrows indicating that a constraint is implied by another constraint *also* indicate that the constraint pointed to is more general. This hierarchy reflects the intended meaning of each constraint in the hierarchy, and the pattern matcher is designed to reflect these meanings (i.e., the pattern matcher will match the pattern link "A *touches* B" to the link "X *supports* Y" in an instance). Similar partial orderings are easily defined for allowed constraints on the node features shape and orientation (e.g. Wedge > (Wedge OR Brick) > Not-Sphere).

It is easy to show, given these partial orderings on individual links and node features, that pattern P1 is more specific than P2 if and only if there is a legal mapping of nodes and links in P1 to those in P2 such that each constraint in P1 is more specific than the corresponding constraint in P2.

[1] To be a complete description of the partial ordering on relations, each of the three lower relations in the figure should also point to the constraint "Any relation".

The importance of the general-to-specific ordering of concepts lies in the fact that for a broad class of concept description languages, any version space can be represented in terms of its sets of maximal and minimal patterns according to this ordering. We shall refer to the subset of *maximally specific patterns* of a set of patterns, X, as MAX(X), and to the subset of minimally specific or maximally general patterns of X as MIN(X), where

$$MAX(X) = \{x \in X \mid (\forall y \in X)\ \neg(y > x)\}$$

$$MIN(X) = \{x \in X \mid (\forall y \in X)\ \neg(x > y)\}$$

The boundary sets of ***maximally specific patterns***, (S), and ***maximally general (or minimally specific) patterns***, (G), of a given version space may be used to represent the version space. The concept descriptions contained in the version space are exactly those contained in the sets S and G as well as all concept descriptions between these two sets in the partial ordering.

In the arch-learning example, detailed below, the version space of all concept descriptions consistent with a given training set of arches and non-arches is represented by the sets of maximally specific and maximally general concept descriptions consistent with these instances.

This ***boundary sets representation*** for version spaces is both compact and easily updated in response to new training instances. In chapter 3, a proof is given that for a well defined broad class of concept description languages (including any language containing a countably infinite number of concept descriptions), the version space associated with any set of training instances can be correctly represented by its boundary sets S and G. Furthermore, it is proven that the stated method for updating these boundary sets in response to new training instances yields the representation for the correctly updated version space.

2.3.3 Using Version Spaces for Concept Learning

Version spaces are the data structures with which the learning algorithm works. This algorithm, called the ***candidate elimination algorithm***, begins by initializing the version space to the set of all concept descriptions consistent with the first observed positive training instance. Subsequent training instances are then considered one at a time, and any concept descriptions inconsistent with the current training instance are eliminated from the version space. Thus, in the end only concept descriptions consistent with all the training instances remain in the version space.

Although we speak of the candidate elimination algorithm as operating on the version space, it actually operates on the boundary sets, S and G, which *represent* the version space. Below we explain the candidate elimination algorithm by tracing its behavior in learning the arch concept [1].

2.3.3.1 Initializing the Version Space

The version space is initialized to the set of all concept descriptions consistent with the first positive training instance. This is accomplished by initializing the sets S and G, respectively, to the maximally specific and maximally general concept descriptions which match the first positive training instance.

[1] The following is a hand simulation - not a computer implementation - of the version space approach to this problem.

The most specific concept description which matches the first positive training instance of an arch, as shown in figure 2.2, is the description of the instance itself. This is the most specific (most constrained) concept description within the search space which matches the training instance, and therefore is taken as the initial element of the set S. The G boundary of the version space is initialized to the most general concept description which matches the instance. In this case, G contains the general concept description which states that an arch is any collection of three objects with any properties and any relations among these three objects (recall that in this example a pattern can match only instances containing the same number of nodes as the pattern).

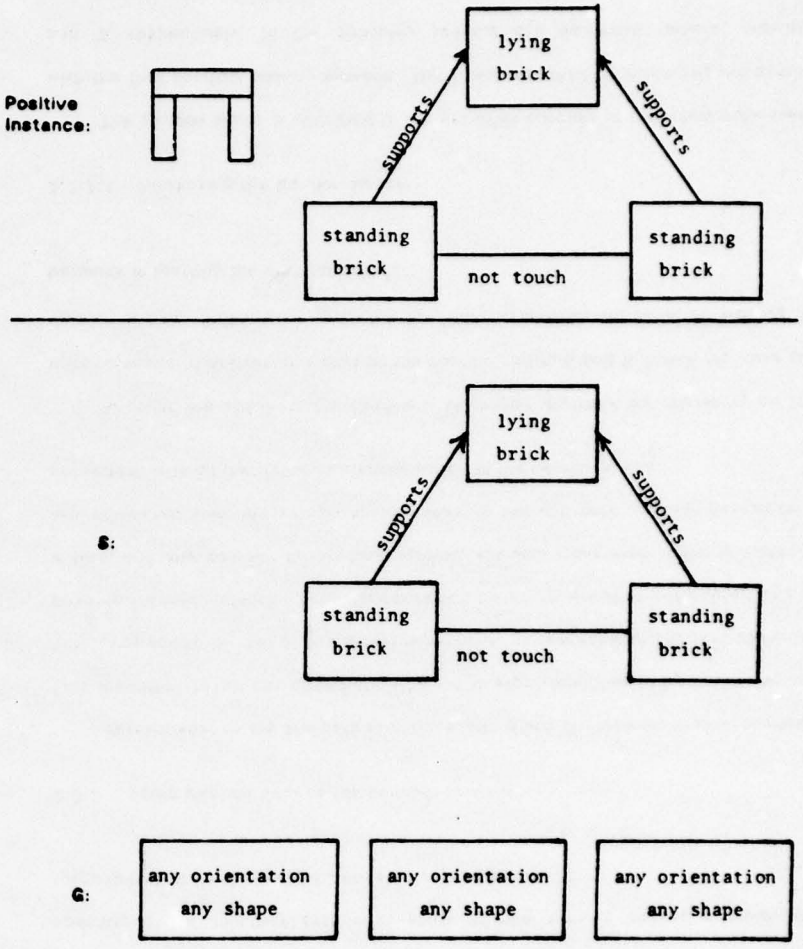


Figure 2.2 Initializing the version space.

Notice that there are very many candidate versions of the target concept contained in this initial version space: namely the members of S and G as well as all concept descriptions which lie between these two boundary sets in the partial ordering. It is clear to us that the current G is too general, but on the basis of the single observed training instance the program has not yet discovered this fact. Further training instances will reduce the number of concept descriptions contained in the version space, moving the boundary sets S and G toward each other in the partial ordering. Notice, however, that the version space can never contain any candidate version of the target concept which it does not contain at this point. Any such concept description is inconsistent with this first training instance.

2.3.3.2 Processing Training Instances

Once the version space is initialized, subsequent training instances are considered one at a time. At each step, the sets S and G are modified as needed to eliminate from the version space those concept descriptions which are inconsistent with the current training instance. These boundary sets of the version space move monotonically toward each other as they converge toward the target concept. Figure 2.3 depicts this behavior. The entire figure represents the search space or language of concept descriptions. Specific patterns are located toward the top of the box, and more general patterns toward the bottom, as indicated by the arrow at the left. The boundary sets of maximally specific and maximally general concept descriptions delimit the current version space within the search space.

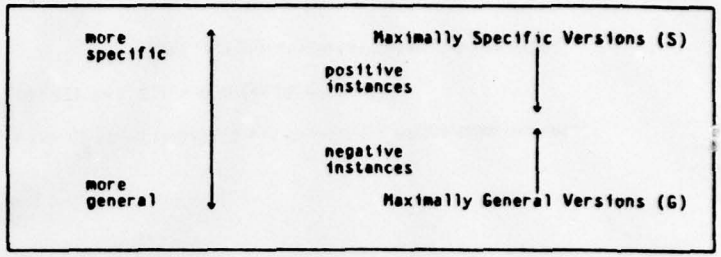


Figure 2.3

Effect of Positive and Negative
Training Instances on Version Space Boundaries

The movement of the S boundary of the version space may be viewed as a breadth first search from specific to general patterns. The boundary is forced to become more general as new positive instances are encountered which are not matched by all patterns in S. The breadth and depth of this search are controlled by the requirement that each pattern in S be more specific than some pattern in G (this assures that each S pattern is consistent with all past negative instances).

In a complementary fashion, the G boundary moves in breadth first manner from general to specific descriptions of the target concept. It is forced to become more specific as new negative instances are matched by some pattern in G. The breadth and depth of this search are controlled by the requirement that each pattern in G be more general than some pattern in S (this assures that each G pattern is consistent with all past positive instances).

The interplay between these two complementary searches is simple, but crucial to controlling the combinatorics of the problem. Requiring patterns in each boundary set to lie along the same branch of the partial ordering as some pattern in the opposite boundary keeps both searches from getting too broad or deep. These constraints the two boundary sets impose on each other are not heuristic or approximate - they are exact constraints justified by the past training instances. In a sense, the S boundary provides a summary (in the language of concept descriptions) of all past positive instances. It tells which additional constraints may be added to the general patterns in the G boundary. This summary is used to direct the search traced out by the G boundary which, in turn, summarizes the negative instances and is used to direct the S search.

The algorithm for updating the boundary sets in response to a training instance is described below. In the formal notation, the predicate M is the pattern matching predicate, the set P is the set of all concept descriptions in the prescribed language, the set VS is the current version space, and the functions MIN and MAX are as described earlier in this section.

The Candidate Elimination Algorithm

IF new instance, I, is a negative instance,

  THEN BEGIN

    COMMENT Retain in the set S, only patterns which do not match i;

    $S \leftarrow \{s \in S \mid \neg M(s,I)\}$;

    COMMENT Replace G by the set of least specific patterns in the current version space which do not match I;

    G ← UPDATE-G(G, S, I);

  END

ELSE IF i is a positive instance,

  THEN BEGIN

    COMMENT Retain in the set G, only patterns which match i;

    $G \leftarrow \{g \in G \mid M(g,I)\}$;

    COMMENT Replace S by the set of most specific patterns in the current version space which match I;

    S ← UPDATE-S(S, G, i);

  END.

Where the functions UPDATE-G and UPDATE-S yield the following sets.

$$\text{UPDATE-G}(G, S, I) = \text{MIN}(\{p \in VS \mid \neg M(p,i)\})$$

$$= \text{MIN}(\{p \in P \mid (\exists s \in S)(\exists g \in G)((s \geq p \geq g) \wedge \neg M(p,I))\})$$

$$\text{UPDATE-S}(S, G, I) = \text{MAX}(\{p \in VS \mid M(p,I)\})$$

$$= \text{MAX}(\{p \in P \mid (\exists s \in S)(\exists g \in G)((s \geq p \geq P) \wedge M(p,I))\}).$$

The functions UPDATE-G, UPDATE-S,[1] MIN, and MAX, and the predicate M must all, of course, depend upon the particular concept description language. With these procedures correctly defined, the algorithm as stated above is valid for any concept learning problem for which the sets G and S correctly represent the version space. Chapter 3 contains a proof of the correctness of this algorithm, as well as conditions under which the boundary sets S and G may be used to represent the version space.

Consider how this algorithm applies to the arch learning example. When the second training instance, this time a negative instance, is encountered, the version space is updated as shown in figure 2.4. Here, the pattern contained in the S boundary is consistent with the negative instance (it correctly does not match the instance) and S is therefore left unaltered.

On the other hand, the pattern in the G boundary of the version space is now found to be too general. It incorrectly matches the negative instance, predicting that this instance is an arch. Thus the pattern contained in G is eliminated from the version space, and the G boundary is moved to the set of least more specific patterns which do not match the negative instance, yet which remain more general than the S element. In this case, the *touches* relation is added to the old G pattern to arrive at the new G boundary of the version space.

[1] Functions similar to UPDATE-S have been implemented by several researchers for particular pattern languages (e.g. the least generalization algorithm of Plotkin [Plotkin, 1970], the interference matching algorithm of Hayes-Roth [Hayes-Roth, 1975], and the maximal common generalizations algorithm of Vere [Vere, 1975]).

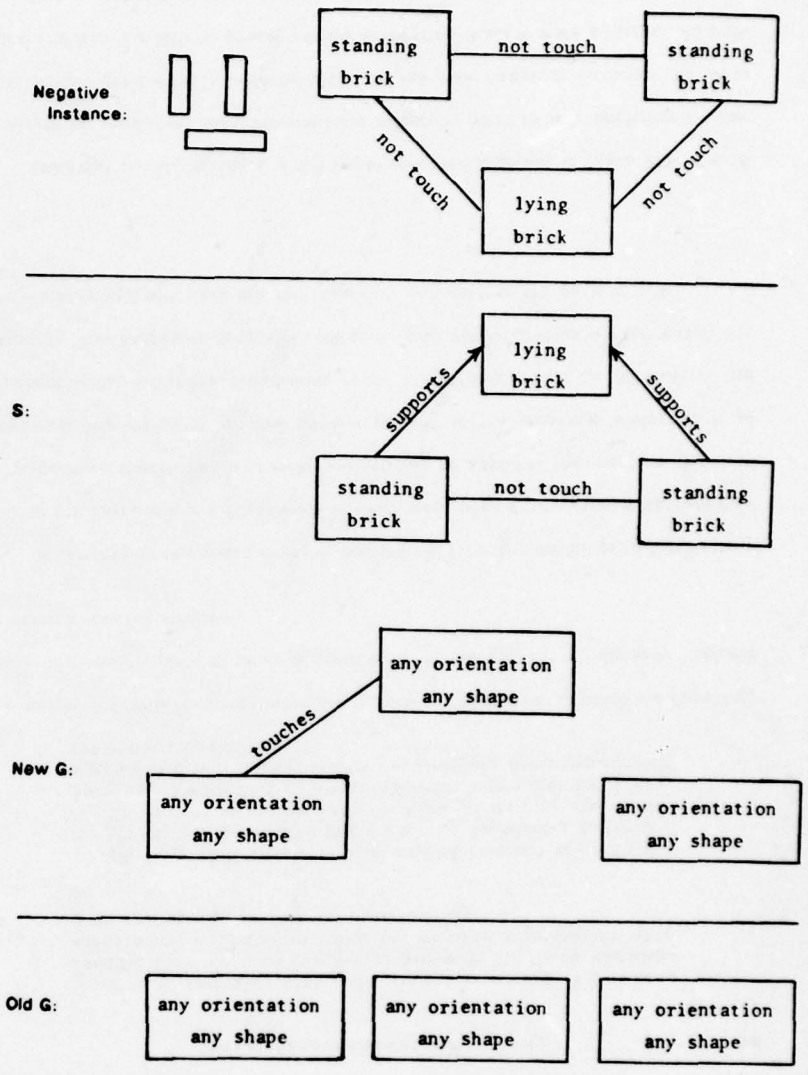


Figure 2.4 Revising the version space on the basis of a negative training instance.

It is important to understand why the *touches* relation is added to arrive at the new G, while alternate constraints (alternate branches in the search) are not considered. The *supports* relation was not considered since, as mentioned above, *supports* is a more specific constraint than *touches*. Since the *touches* relation is sufficient, adding the *supports* relation would not yield a maximally general concept description.

A different constraint which might have been added to the old G to disallow the match to the negative instance would be to constrain the shape of one of the objects to be something other than a brick. This branch of the search is not considered because the resulting pattern is no longer more general than S (and is therefore inconsistent with some past positive instance!). Thus, although there are many patterns more specific than the old G which do not match the new negative instance, in this case there is only one maximally general such pattern consistent with S. In general, of course, there may be several patterns in the updated boundary set G.

The procedure UPDATE-G(G, S, i), which determines the patterns minimally more specific than G, less specific than S, and which do not match the negative instance i, must of course depend upon the pattern language. For the current language, such a procedure could be written using the fact that patterns are made more specific only by tightening (making more specific) existing constraints on individual nodes or links, or by adding new constraints. Given a pattern g, which matches i, one procedure for finding all minimally more specific patterns less specific than some pattern in S, which do not match i, is the following:

1) For each pattern, s, in S, find all mappings of nodes in g to those in s, such that each node and link constraint in g is less specific than the corresponding constraint in s. These mappings indicate how each constraint in g may be made more specific while g remains more general than or equal to s.

2) Consider in turn, each legal mapping of nodes and links in g to those in i, corresponding to the mappings for which g matches i. Find the set of all minimal ways (of those allowed by step 1) for making g more specific, such that this mapping no longer constitutes a match of g to i. Repeat this for each mapping, using the updated set of patterns at each step.

The procedure UPDATE-G may therefore replace any pattern in G which matches i by the set derived from the above procedure, then find the set of minimally specific patterns in the resulting set.

In processing the above negative training instance, the mappings of the pattern in G to the pattern in S indicate ways in which each node in the general pattern may be made more specific, as well as links which may be added to that pattern. By then examining the mappings of this general pattern to the negative instance, it is apparent which of these possibilities result in patterns which do not match the instance. This procedure results in considering only patterns which will not match the new instance and which are assured to be more general than the pattern in S.

The next training instance is the negative instance shown in figure 2.6. The S boundary is consistent with the negative instance, so it is left unchanged. The pattern contained in G incorrectly matches the new negative instance, so it is eliminated from the version space, and the G boundary is made more specific. As with the previous training instance, although there are many sets of constraints which

could be added to the old G to disallow its match to the negative instance, there is only a single resulting pattern in the new G boundary set. Any other way of making the G pattern more specific would be more specific than the new G shown in the figure, would fail to disallow the match to the negative instance, or would fail to remain more general than the S version space boundary.

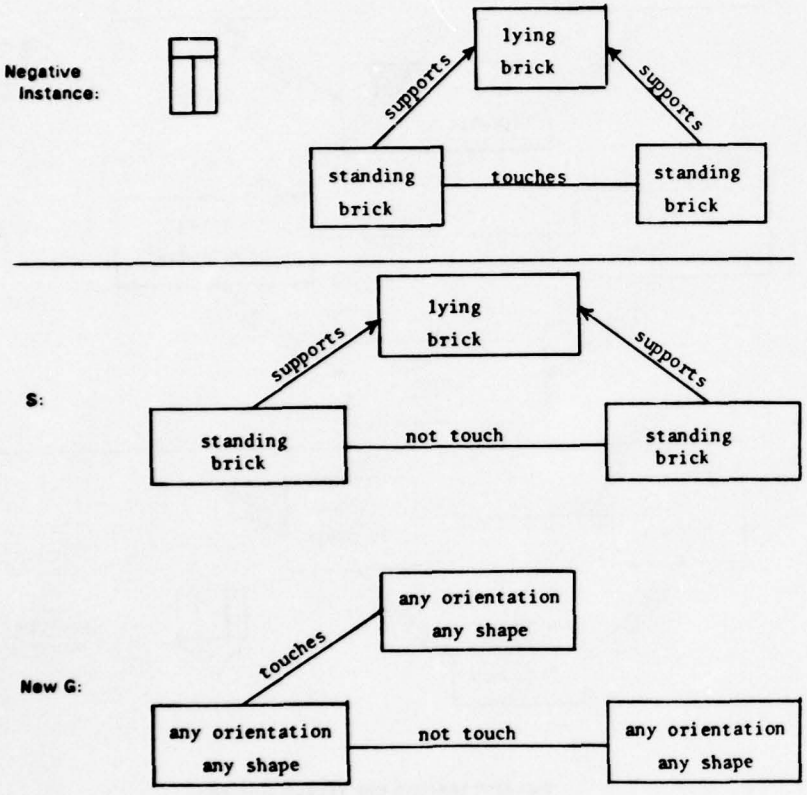


Figure 2.5

Figure 2.6 shows the effect on the version space of a new positive training instance. In this case the G boundary of the version space correctly matches the positive instance, and is therefore left unchanged. The pattern contained in the S boundary does not match the new instance, so it is eliminated from the version space, and the S boundary moved to the set of minimally more general patterns (still more specific than G) which match the instance. As with updating G in previous instances, there are many ways of making the pattern in S more general to make it consistent with the new instance, but in this case, there is only a single pattern contained in the resulting S.

The procedure UPDATE-S(S, G, i), which determines all patterns minimally more general than those in S, which are more specific than some pattern in G, and which match the instance i, is used to obtain the new set S for this instance. One strategy for implementing UPDATE-S is to apply the following procedure to each pattern in the set S, then determine the set of maximally specific resulting patterns.

> 1) For each pattern, s, in the set S, determine all mappings of nodes in s to those in i.
>
> 2) For each mapping found in step 1, generalize each node and link constraint in s as needed until it matches the corresponding feature value or link in i. This yields the minimally more specific pattern which matches i under this mapping. Find the maximally specific patterns in the resulting set, and delete those which are not more specific than some pattern in the set G.

In the example of figure 2.6, the obvious mapping of the pattern in S to the new positive instance is the mapping which results in the maximally specific resulting pattern shown in the figure.

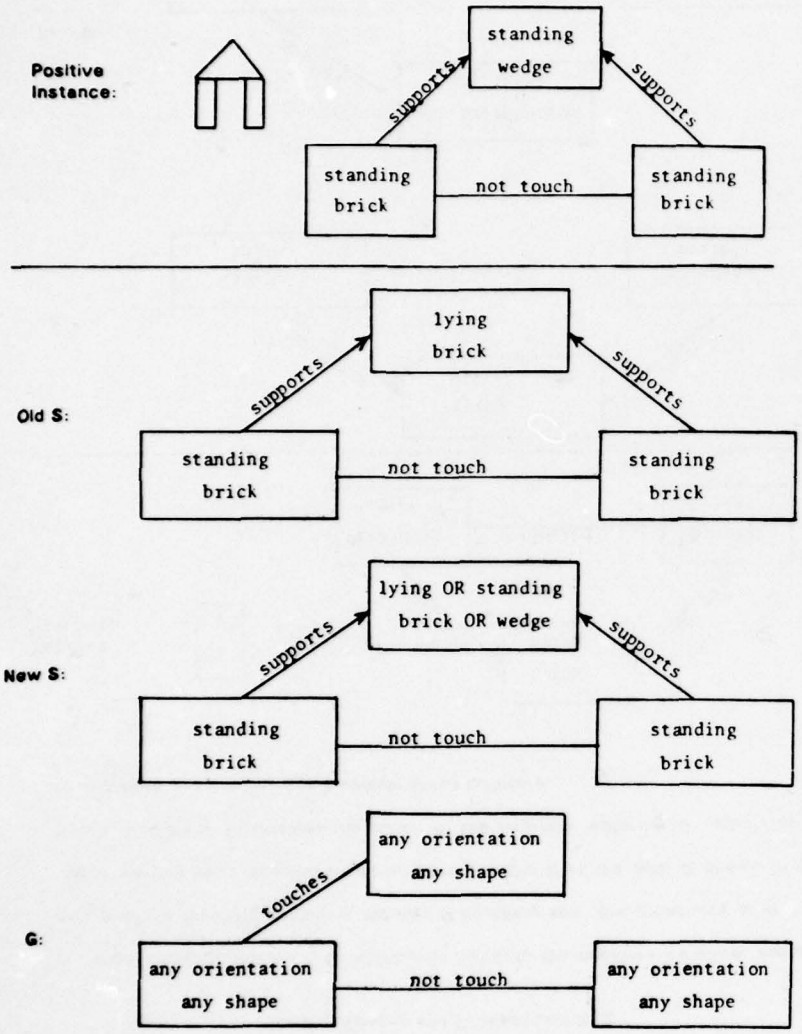


Figure 2.6

The four training instances considered thus far are exactly the four used by Winston [Winston, 1975] to teach the arch concept to his program. The version space delimited by the S and G boundaries in figure 2.6 contains all the concept descriptions (within the described language) which are consistent with these four training instances. From the number of alternate plausible concept descriptions in the version space it is clear that the information contained in these four training instances is not sufficient to unambiguously describe the arch concept. This point illustrates an important feature of the version space approach to concept learning: by examining the alternate concept descriptions in the version space, it is possible to recognize whether enough training data has been processed to unambiguously learn the concept. This point is discussed further in chapter 4.

Figures 2.7 and 2.8 illustrate two additional training instances (chosen by the author) and their effects on the version space of the arch concept.

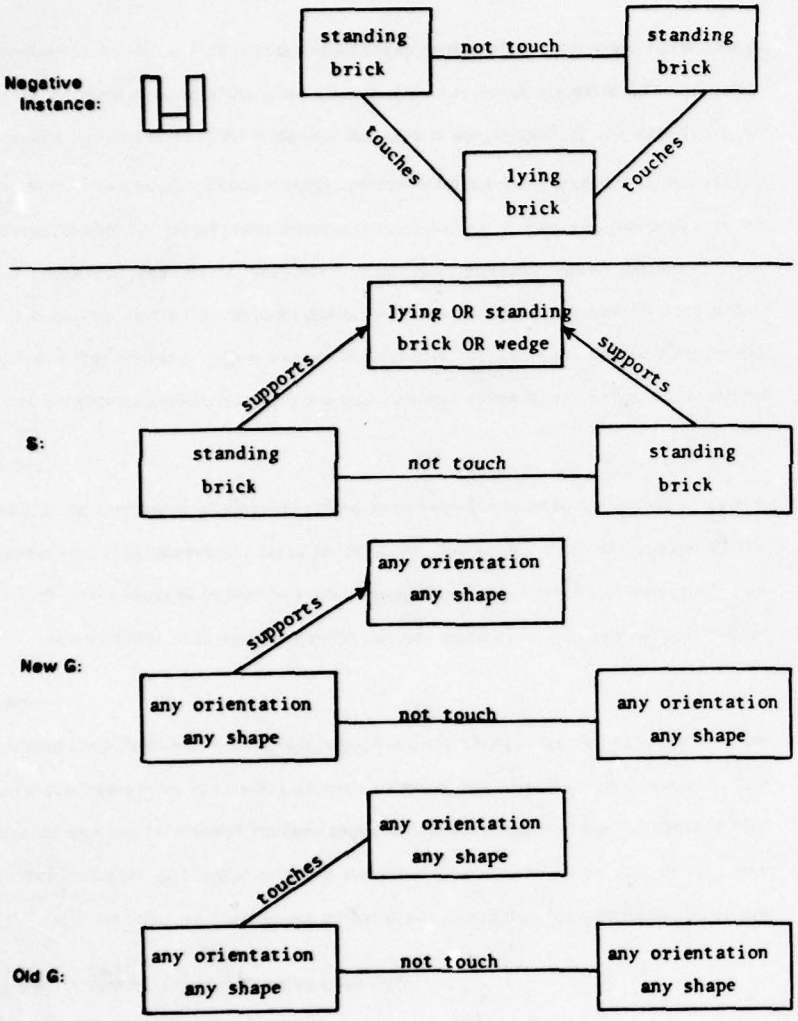


Figure 2.7

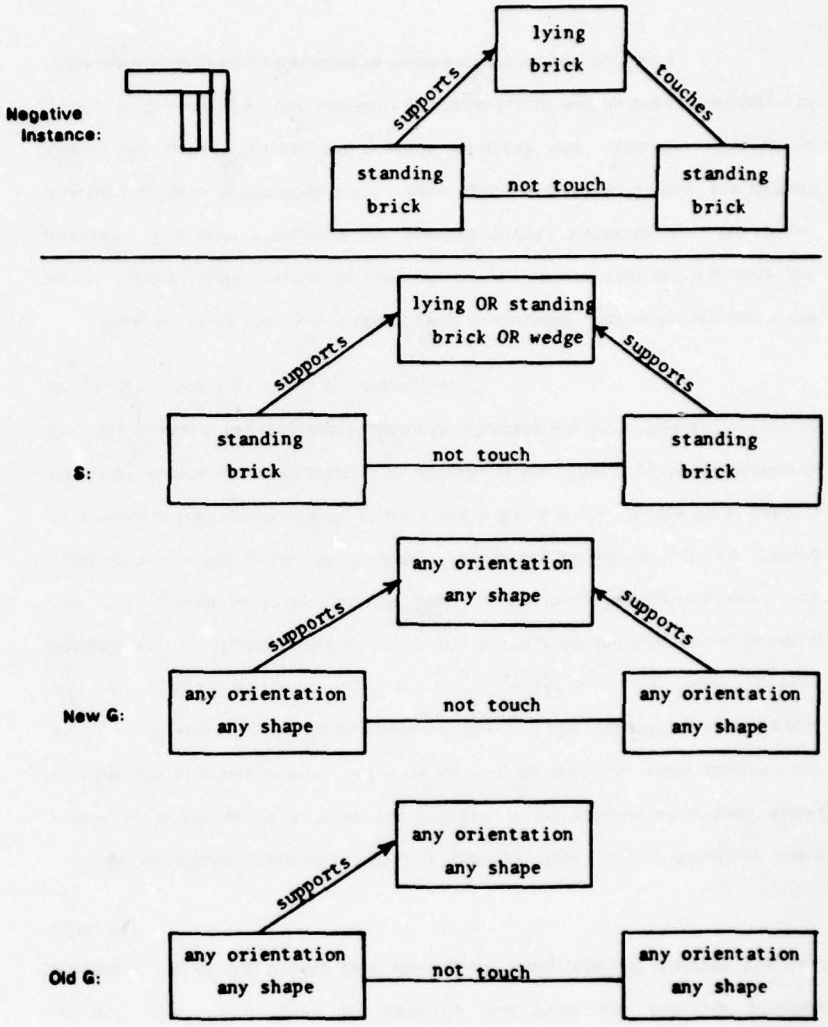


Figure 2.8

2.3.4 Applying Partially Learned Concepts

At some point the program will be expected to apply its learned definition of the target concept. Depending upon the training data observed up to that point, there *may or may not be a single concept description in the version space. If there is not,* then the identity of the target concept has only been partially determined by the training instances, and some strategy for using the partially learned concept must be found.

One strategy is to choose a single concept description from the version space, relying upon heuristics or outside knowledge of the domain to make this selection. The chosen concept description could be taken as the current best hypothesis of the identity of the target concept, and used to classify future possible instances of the concept.

A second strategy is to take the entire version space as the current description of the target concept. This is a more complete strategy, since the patterns contained in the version space *represent all alternate interpretations of the training data within* the concept description language. With this strategy, new instances are characterized by taking each pattern in the version space into account. If all patterns in the version space classify the new instance as an example of the target concept[1], then the program is certain that this is an instance of the concept (i.e., *every possible interpretation of the training instances yields the same classification*). However, if some patterns in the version space match the new instance while others

[1] This may be determined simply by testing elements of S against the new instance.

do not, then the program has detected that there are alternate possible interpretations of the training data yielding conflicting interpretations of the new instance.

As an example, suppose that only the first four training instances were presented in the above arch learning problem. If the program were then asked whether the structure in figure 2.7 were an arch, its response would depend upon which of the above two strategies were employed. If the first strategy were used, then the program *would answer "yes" or "no", depending upon which concept* description were chosen from the version space. If the second strategy were used, then the answer would be "On the basis of the observed training data, the classification is ambiguous. Some concept descriptions consistent with the training data indicate that this is an arch, others indicate that it is not." On the other hand, if the new instance were classified as an instance of the concept by every pattern in the version space, the response would be "All alternate interpretations of the training data indicate this is an instance of the concept."

Thus, by using the version space itself to describe what is known about the target concept, it is possible to make reliable use of whatever training data are available. The more informative the provided training instances are, the fewer alternate concept descriptions in the version space, and the more often the version space will classify subsequent possible instances with certainty. Chapter 4 discusses this use of version spaces in greater detail, as well as the related issue of choosing informative new instances to refine partially learned concepts.

2.3.4.1 Efficiency

Section 2.2.2 discussed how each boundary set of the version space is used to limit the size of the other. This effect was apparent in the above example in which the S and G boundary sets never contained more than a single concept description. These sets will in general contain multiple patterns, as illustrated in the following section, since the general-to-specific ordering is a partial ordering. Thus, even though the boundary sets representation for version spaces constitutes a major step toward containing the combinatorial explosion inherent to learning, the explosion is not smothered completely.

The size of the boundary sets is not directly related to the number of concept descriptions in the version space, as is clear from the above example. The size of these sets is, however, sensitive to the the particular training instances encountered. Although the first four training instances in the above arch learning example were not chosen with the version space approach in mind (they were taken directly from [Winston, 1975]), each was carefully chosen by Winston to highlight a single important aspect of the arch concept. This strategy for choosing training instances helps control the size of the version space boundaries by limiting the branching in the breadth first searches. *Presumably, these training instances also limited the* branching in Winston's depth first search strategy.

The problem of large boundary sets is more evident in more complex search spaces. In the Meta-DENDRAL program (see chapter 6), boundary sets containing 100 patterns have at times been required to represent the version spaces of

chemical concepts learned by that program. In such cases the concept descriptions contained dozens of nodes and links in contrast with the three node arch example.

Although the size of the boundary sets representing version spaces can be a serious problem, it can be controlled to some degree by careful selection of training instances, and of course, by introducing heuristic pruning[1]. Chapter 4 discusses strategies for generating training instances to control the size of the S and G boundaries. Chapter 7 contains a further discussion of relative efficiency of the *version space approach and several other concept learning strategies.*

2.4 A Second Example: Learning Feature Value Intervals

Here we briefly consider a second concept learning problem in order to provide a different perspective on the version space approach. This problem involves learning concepts from training instances represented as feature vectors. The task is to determine ranges of allowed numerical values of a given set of features which distinguish positive instances of the concept from negative instances. The feature *value interval learning problem outlined here is similar* to that studied by [Brown, 1977] in which the problem of learning bridge bidding strategies was considered.

This is a simpler problem than learning network or structural descriptions of concepts; it involves no relations between objects. In fact, the kind of learning described in this example - determining possible generalizations of properties of individual objects - might take place for each node in a structural description learning

[1] Introducing heuristic pruning, however, invalidates the guarantee that *all* concept descriptions consistent with the data will be found.

problem. Because of its relative simplicity this problem provides an illuminating glimpse of the version space approach to forming generalizations.

Described in the format used earlier, the interval learning problem is:

Interval Learning Problem:

Given: *1. Concept Description Language:* All pairs of intervals of the following form where a and b are real numbers.

$a < x < b$
or $a \leq x < b$
or $a < x \leq b$
or $a \leq x \leq b$

For example, the following is a legal concept description:

$(2<x\leq 5,\ 10\leq y<500)$.

2. Pattern Matcher: A pattern matches an instance if each feature value in the instance lies within the corresponding interval specified by the pattern (the positive instance below matches the above pattern, while the negative instance below does not).

3. Training instances: Ordered pairs of values of the properties X and Y, along with an assignment of the training instance as a positive or a negative instance of the target concept.

For example, the following are training instances:

(4 , 300) - positive instance
(6 , 200) - negative instance

Determine:

All concept descriptions within the given language which are consistent with every training instance.

The above definition of pattern matching for this concept description language, taken together with the earlier general definition of "more specific than or equal to" leads to the following partial ordering on patterns in this example.

Partial Ordering for Feature Value Intervals: Pattern P1 is *more specific than or equal to* pattern P2 if each interval specified by P1 is contained in the corresponding interval specified by P2.

For example, the pattern $(2<x<6,\ 5<y<10)$ is more specific than the pattern $(0<x<6,\ 3<y<12)$.

Figure 2.9 illustrates the sequence of version spaces (generated by program) which result from applying the candidate elimination algorithm to the sequence of training instances shown.

The process begins by initializing the version space to the space of all patterns consistent with the first positive training instance, in this case (4, 8). The most specific pattern consistent with this positive training instance (i.e., the pattern which can match *only* this instance) is used to initialize the set S. The most general pattern in the language (which matches *every possible instance*) is clearly consistent with the first positive training instance, and is therefore used to initialize the set G. The two patterns contained in S and G, as well as all patterns between these two in the general-to-specific partial ordering, are all the concept descriptions in the allowed language consistent with this first training instance. Although there are an uncountably infinite number of patterns in the version space, only these two patterns belonging to S and G are needed to represent the version space.

Sequence of Training Instances	Version Space: Specific Boundary (S)	General Boundary (G)
(4, 8) – positive instance		
	(4≤x≤4, 8≤y≤8)	(−∞<x<∞, −∞<y<∞)
(3, 12)– negative instance		
	•	(3<x<∞, −∞<y<∞) (−∞<x<∞, −∞<y<12)
(2, 9) – positive instance		
	(2≤x≤4, 8≤y≤9)	(−∞<x<∞, −∞<y<12)
(1, 9) – negative instance		
	•	(1<x<∞, −∞<y<12)
(5, 7) – negative instance		
	•	(1<x<5, −∞<y<12) (1<x<∞, 7<y<12)
(5, 9) – positive instance		
	(2≤x≤5, 8≤y≤9)	(1<x<∞, 7<y<12)

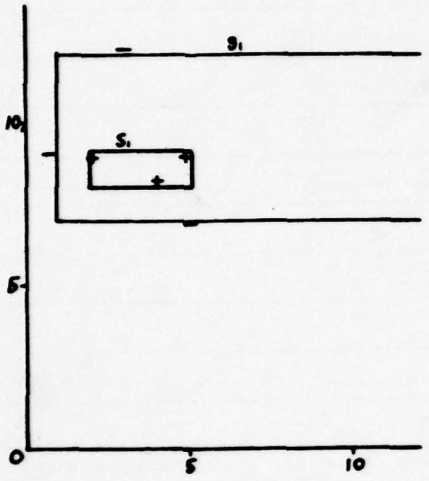
Figure 2.9: Feature Interval Learning Example

Once the version space is initialized, additional training instances are examined one at a time, and versions of the emerging concept which are inconsistent with the current training instance are eliminated from the version space by altering the boundaries S and G. As in the earlier example, negative instances force the G boundary of the version space to become more specific, whereas positive instances force the S boundary to become more general.

The boundary G of the initial version space is inconsistant with the second training instance (it incorrectly matches this negative instance). The boundary is therefore moved to the set of least more specific patterns which do not match the current negative instance and which remain more general than the current S. In this case there are two such patterns lying along different branches in the partial ordering. Notice that in addition to the two patterns shown in the new G, the pattern (−∞<x<3, −∞<y<∞) also is minimally more specific than the old G without matching the new negative instance. Here, however, the constraints imposed by the S boundary on branching of the G boundary come into play. This pattern is not considered since it is not more general than the S boundary, and is therefore known to be inconsistent with some previous training instance.

The resulting version space is pruned further by the third training instance which affects both the S and G version space boundaries. This instance causes S to become more general and eliminates one of the two patterns from G. Notice that a if a pattern contained in G does not match a given positive instance (as is the case here), that entire branch of patterns in the partial ordering is eliminated from the version space. This is because the offending pattern can neither be made more general (it is already as general as possible without matching some negative instance) nor more specific (making any pattern more specific cannot result in it matching an instance which it did not previously match).

The process of eliminating candidate concepts from the version space continues until at the end of the training sequence the version space is bounded by the patterns shown. Figure 2.10 illustrates the patterns contained in the final boundary sets derived from the training data. In this figure, positive instances are plotted as "+" points, and negative instances as "-" points. Each pattern is depicted as a rectangle whose boundaries are defined by the stated intervals on x and y. A pattern matches an instance if the instance is contained within the rectangle corresponding to that pattern. It is possible to verify from the figure that the boundary sets S and G derived by the program are, in fact, the *maximally specific (smallest rectangle)* and maximally general (largest rectangle) patterns in the allowed language which are consistent with all training instances.



G boundary set
g1: 1<x<∞, 7<y<12

S boundary set
s1: 2≤x≤5, 8≤y≤9

Figure 2.10 The version space for a feature interval learning problem.

2.5 Summary

The version space approach to concept learning has been described and illustrated. Subsequent chapters provide a formal justification of the approach, and examine extensions of this approach for dealing with errorful data. The use of version spaces to represent and apply partially learned concepts and to generate informative new training instances is considered, and the use of version spaces for concept learning in the Meta-DENDRAL domain of chemical spectroscopy described. Finally, the use of version spaces for concept learning is compared with earlier methods for concept learning.

Chapter 3

Formal Treatment

3.1 Introduction

A formal description and justification of the version space approach to concept learning is presented in this chapter[1]. A class of concept learning problems is defined, and the version space approach to this class of problems presented. The major results presented in this chapter are the following:

1) The correctness of the boundary sets representation for version spaces is proven for a broad class of admissible concept description languages.

2) The correctness of the procedure for updating these boundary sets in response to new training instances is proven.

3) An extended definition of version spaces is given for dealing with inconsistent concept learning problems (discussed further in chapter 6). The correctness of the boundary sets representation for this definition of version spaces is proven, as well as the correctness of the stated procedures for updating these sets in response to subsequent training instances.

3.2 The Concept Learning Problem

Given a set, I, of all possible instances of concepts, a *target concept* in I is defined as an ordered pair, $\langle C_+, C_- \rangle$, of subsets of I where

[1] I am indebted to Lew Creary and Chitoor Srinivasan for useful suggestions which helped to clarify several issues in the following discussion.

$$C_+ \cap C_- = \emptyset .$$

The set C_+ is known as the set of positive instances of the concept, and C_- as the set of negative instances of the concept.

The concept is presented as a sequence of training instances drawn from C_+ and C_-. At any point in this sequence of training instances the sets I_+ and I_- are defined as follows:

$I_+ = \{i \in C_+ \mid i \text{ has been presented as a positive instance}\}$

$I_- = \{i \in C_- \mid i \text{ has been presented as a negative instance}\}$.

A *concept learning problem* is an ordered triple $\langle P, M, \langle I_+, I_- \rangle \rangle$ where

P = a set of patterns

$M(p, i)$ = a matching predicate where $p \in P$ and $i \in I$

$\langle I_+, I_- \rangle$ = sets of positive and negative instances of the target concept.

We define the consistency predicate K, as follows:

$$K(p, \langle I_+, I_- \rangle) \Leftrightarrow (((\forall i \in I_+)\, M(p,i)) \wedge ((\forall i \in I_-)\, \neg M(p,i)))$$

and will say that a pattern p is *consistent* with training instances $\langle I_+, I_- \rangle$ if and only if $K(p, \langle I_+, I_- \rangle)$.

Given a concept learning problem $\langle P, M, \langle I_+, I_- \rangle \rangle$, the goal of the learner is to find patterns from P which are consistent with the observed training instances of the concept[1]. Such patterns allow distinguishing elements of I_+ from elements of I_-, and

[1] Other criteria, such as "generality" or "elegance", may also be important in choosing a pattern to describe the concept. Here only the central criterion for acceptance, consistency with the training instances, is considered.

are to be used to classify subsequent elements of I. The solution to the concept *learning* problem is the set VS of all such patterns.

$$VS = \{p \in P \mid K(p, \langle I_+, I_- \rangle)\}$$

The set VS associated with a particular concept learning problem $\langle P, M, \langle I_+, I_- \rangle\rangle$ will be called the *version space* of the training instances $\langle I_+, I_- \rangle$ with respect to P and M, or simply the version space associated with the concept learning problem.

3.3 Representing Version Spaces

The version space associated with a particular concept learning problem is the solution to that problem. In order for computers to solve such problems version spaces must be represented efficiently. In this section the *boundary sets representation* of version spaces is discussed.

We begin with a definition of the boundary sets representation for version spaces. This representation utilizes a partial ordering over P which is defined for any concept learning problem by the matching predicate M. The "more specific than or equal to" relation on elements of P is first defined and shown to be a partial ordering. The boundary sets representation is then defined in terms of this partial ordering, and shown to be valid for any pattern language and matching predicate meeting a certain admissibility criterion.

3.3.1 The Partial Ordering

The relation *more specific than or equal to* ($\geq$) is defined on the set of patterns, P, as follows:

$$(\forall p_1, p_2 \in P)((p_1 \geq p_2) \leftrightarrow (\{i \in I \mid M(p_1, i)\} \subseteq \{i \in I \mid M(p_2, i)\})).$$

It will also be useful at times to rely upon the strict relation *more specific than* ($>$) defined as follows:

$$(\forall p_1, p_2 \in P)((p_1 > p_2) \leftrightarrow ((p_1 \geq p_2) \wedge (p_1 \neq p_2))).$$

Theorem 1: Partial Ordering Theorem. The relation $\geq$ on P is a partial ordering.

Proof: The relation is

reflexive: $(\forall p \in P)(p \geq p)$

antisymmetric: $(\forall p_1, p_2 \in P)((p_1 \geq p_2) \wedge (p_2 \geq p_1))$

$\rightarrow (\{i \in I \mid M(p_1, i)\} \subseteq \{i \in I \mid M(p_2, i)\} \subseteq \{i \in I \mid M(p_1, i)\})$

$\rightarrow (p_1 = p_2))$.

and transitive: $(\forall p_1, p_2, p_3 \in P)((p_1 \geq p_2) \wedge (p_2 \geq p_3))$

$\rightarrow (\{i \in I \mid M(p_1, i)\} \subseteq \{i \in I \mid M(p_2, i)\} \subseteq \{i \in I \mid M(p_3, i)\})$

$\rightarrow (p_1 \geq p_3))$.

The following definitions will be useful in considering partially ordered subsets of P. For any subset A of P, we refer to the subset of the *maximally specific elements* of A as MAX(A):

$$MAX(A) = \{a \in A \mid (\forall a' \in A)\ \neg(a' > a)\}.$$

Similarly, we refer to the subset of *minimally specific (or maximally general) elements* of A as MIN(A):

$$MIN(A) = \{a \in A \mid (\forall a' \in A)\ \neg(a > a')\}.$$

Finally, we define a *chain* of P as a totally ordered subset of P. That is, a subset A of P is a called *chain* of P if and only if

$$(\forall x, y \in A)(x \geq y \vee y \geq x)$$

3.3.2 The Boundary Sets Representation for Version Spaces

Given a concept learning problem <P, M, $\langle I_+, I_- \rangle$> and the associated version space VS, the *boundary sets representation* for VS is defined as the ordered pair <G, S> of subsets of P, where

$$G = MIN(VS)$$

$$S = MAX(VS).$$

Notice that by the definition of VS,

$$G = MIN(\{p \in P \mid (K(p, \langle I_+, I_- \rangle))\})$$

$$S = MAX(\{p \in P \mid (K(p, \langle I_+, I_- \rangle))\}).$$

The pair of sets <G, S> associated with a concept learning problem is said to represent the version space, VS, associated with that problem if and only if

$$(\forall p \in P)((p \in VS) \Leftrightarrow (\exists g \in G)(\exists s \in S)(s \geq p \geq g)).$$

Theorem 2 states that for admissible pattern languages, the version space associated with any concept learning problem can be represented by its boundary

sets <G, S>. Before presenting this theorem, we define the admissibility criterion for pattern languages.

3.3.3 Admissibility of Pattern Languages

A set of patterns P with associated matching predicate M is said to be an *admissible* pattern language if and only if every chain of P has a maximum and a minimum element.

Notice that since every finite or countably infinite chain has a maximum and a minimum element, *every finite or countably infinite set of patterns yields an admissible pattern language*. Therefore, the structural pattern languages used in the arch learning problem of chapter 2 and the chemistry problem of chapter 6 are admissible.

An example of an inadmissible pattern language helps further clarify the admissibility criterion. Consider the language of patterns described in the second example of chapter 2 (section 2.4). Here P is a language of feature value intervals, consisting of patterns such as $(3<x<6,\ 4\leq y<6)$. This language is admissible even though it is uncountably infinite. The language becomes inadmissible, however, if only the strict "<" relation is allowed in defining feature value intervals, and not the relation "≤". The resulting language is inadmissible, and the version space associated with any set of training instances cannot be represented by its boundary sets. Consider, for example, the positive training instance (4, 8). The pattern $(3.99<x<4.01,\ 7.99<y<8.01)$ is consistent with this training instance, and there is an infinite chain of progressively more specific patterns also consistent with the instance, with no maximum element.

3.3.4 Validity of the Boundary Sets Representation for Version Spaces

For concept learning problems involving admissible pattern languages, version spaces may be represented in terms of their boundary sets ⟨G, S⟩.

Theorem 2: Validity of the boundary sets representation.

Consider a concept learning problem ⟨P, M, ⟨I_+, I_-⟩⟩, with associated version space VS, and associated boundary sets ⟨G, S⟩, and for which I_+ and I_- are non-empty.

If P and M are admissible, then $(\forall p \in P)((p \in VS) \leftrightarrow (\exists g \in G)(\exists s \in S)\ (s \geq p \geq g))$.

Proof: We first prove that $(\forall p \in P)((p \in VS) \leftarrow (\exists g \in G)(\exists s \in S)\ (s \geq p \geq g))$.

Consider arbitrarily chosen $p \in P$ and $s \in S$ such that $s \geq p$. By $S \subseteq VS$, we have

$$(\forall i \in I_+)(M(s,i))$$

and by $s \geq p$,

$$\{i \in I \mid M(s,i)\} \subseteq \{i \in I \mid M(p,i)\} .$$

Thus, $(\forall i \in I_+)(M(p,i))$. (1)

Similarly, for arbitrarily chosen $g \in G$ such that $p \geq g$,

$$(\forall i \in I_-)(\neg M(g,i))$$

and by $p \geq g$,

$$\{i \in I \mid M(p,i)\} \subseteq \{i \in I \mid M(g,i)\} .$$

Taking the complement of each set,

$$\{i \in I \mid \neg M(p,i)\} \supseteq \{i \in I \mid \neg M(g,i)\} .$$

Thus, $(\forall i \in I_-)(\neg M(p,i))$. (2)

By (1) and (2), $p \in VS$.

The proof that $(\forall p \in P)((p \in VS) \rightarrow (\exists g \in G)(\exists s \in S)\ (s \geq p \geq g))$ follows directly from the admissibility of P and M. Consider an arbitrary $p \in VS$, and chose a maximal chain, A, of VS that contains p. Because P and M are admissible, A contains a maximum element, s. Because A is a maximal chain of VS, $s \in S$. Similarly, A has a minimum element which is contained in the boundary set G. Thus,

$$(p \in VS) \rightarrow (\exists s \in S)(\exists g \in G)(s \geq p \geq g),$$

and the theorem is proven.

Theorem 2 shows that for the class of admissible pattern languages, one can determine whether a pattern belongs to a given version space by a simple test involving the members of G and S. Thus the procedure for testing for membership of an arbitrary pattern in a given version space is efficient with this boundary sets representation of version spaces.

3.4 The Concept Learning Algorithm

In this section, a proof is given of the validity of the procedure described in chapter 2 for revising the boundary sets ⟨G, S⟩ in response to a new training instance. We consider two learning problems which differ by a single training instance, and show that the boundary sets associated with the second concept learning problem may be derived from those associated with the first problem and the new training instance. Theorem 3 covers the case in which the new training instance is a negative instance, while theorem 4 covers the case where the new instance is a positive instance.

Theorem 3: Revising boundary sets given a negative training instance.

Consider a concept learning problem CP=⟨P, M, ⟨I_+, I_-⟩⟩ with associated VS, S, and G; and a second concept learning problem CP' = ⟨P, M, ⟨I_+, I_-'⟩⟩ with associated VS', S', and G', where I_-' = I_- ∪ {i}. Then the sets S' and G' obey the following equalities.

$$G' = MIN(\{p \in VS \mid \neg M(p,i)\})$$

$$S' = \{s \in S \mid \neg M(s,i)\}.$$

Proof: By definition of the consistency predicate, we find that

$$K(p, \langle I_+, I_-' \rangle) \Leftrightarrow (K(p, \langle I_+, I_- \rangle) \wedge \neg M(p,i))$$

and by definition of VS,

$$VS' = \{p \in P \mid K(p, \langle I_+, I_-' \rangle)\}$$

$$= \{p \in P \mid K(p, \langle I_+, I_- \rangle) \wedge \neg M(p,i)\}$$

$$= \{p \in P \mid (p \in VS) \wedge \neg M(p,i)\}$$

$$= \{p \in VS \mid \neg M(p,i)\}.$$

Therefore,

$$G' = MIN(VS') = MIN(\{p \in VS \mid \neg M(p,i)\}).$$

The first half of the theorem is proven.

In order to prove S' = {s∈S| ¬M(s,i)}, we first prove that S'⊆S.

Consider arbitrary s'∈S'. We wish to prove that s'∈S. We begin with

$$(\forall p \in VS)\ ((p \in VS') \vee (p \notin VS'))$$

By definition of S', we know that (∀p∈VS') (¬(p>s')). Therefore,

$$(\forall p \in VS)\ (\neg(p > s') \vee (p \notin VS')).$$

Since VS' = {p∈VS| ¬M(p,i)}, we know that

$$(\forall p \in VS)((p \notin VS') \rightarrow M(p,i) \rightarrow \neg(p > s'))$$

Thus (∀p∈VS)(¬(p>s')). But S = MAX(VS) = {q∈VS| (∀p∈VS)(¬(p>q))}. Therefore, s'∈S, and we have proven that S'⊆S.

Since S' = MAX(VS') = MAX({p∈VS| ¬M(p,i)}), and S'⊆S⊆VS, we have

$$S' = \{p \in S \mid \neg M(p,i)\},$$

and the theorem is proven.

Theorem 4: Revising boundary sets given a positive training instance.

Consider a concept learning problem CP = ⟨P, M, ⟨I_+, I_-⟩⟩, with associated VS, S, and G; and a second concept learning problem CP= = ⟨P, M, ⟨I_+', I_-⟩⟩ with associated version space VS' and boundary sets S' and G'. Assume I_+' =I_+ ∪ {i}. The sets S' and G' associated with CP' obey the following equalities:

$$S' = MAX(\{p \in VS \mid M(p,i)\})$$

$$G' = \{g \in G \mid M(g,i)\}.$$

Proof: This theorem is the dual of theorem 3. Its proof is analogous to the proof of that theorem.

3.5 Inconsistent Learning Problems

This section presents an extension of the version space approach which allows dealing with inconsistent concept learning problems; that is, concept learning problems for which *no* pattern from P is consistent with the training instances. This extended approach deals with inconsistency by considering sets of patterns consistent with

subsets of the training instances. Methods for dealing with inconsistent learning problems are discussed in detail in chapter 5. This section provides the theoretical foundation for that discussion.

A more general definition of the term version space is introduced, in which a version space contains patterns consistent with at least one of a set of alternate pairs $\langle I_+, I_- \rangle$. The validity of representing such version spaces by their boundary sets is proven in theorem 5. Theorems 6 and 7 prove the correctness of the procedure described in chapter 5 for revising these boundary sets.

We begin by introducing the following notation which will be useful in referring to sets of subsets of I_+ and I_-. We denote the set of m-element subsets of A by SS(A, m). PS(A) refers to the power set of A.

$$SS(A, m) = \{X \in PS(A) \mid (\text{cardinality of } X) = m\}.$$

In the following, unless otherwise stated, when discussing a concept learning problem $\langle P, M, \langle I_+, I_- \rangle \rangle$, the symbols pos and neg are taken to be the cardinality of I_+ and I_- respectively.

$$pos = \text{cardinality of } I_+$$

$$neg = \text{cardinality of } I_-.$$

In inconsistent learning problems, there is no pattern consistent with all training instances. Chapter 5 describes strategies for determining patterns consistent with the largest possible subsets of training instances. To accomplish this, it is useful to generalize the definition of version spaces as follows.

We define the version space $VS_{s,g}$ associated with the concept learning problem $\langle P, M, \langle I_+, I_- \rangle \rangle$ as the set of all patterns consistent with *some* pos-s element subset of I_+ and *some* neg-g element subset of I_-.

$$VS_{s,g} = \{p \in P \mid (\exists T_+ \in SS(I_+, pos-s))\ (\exists T_- \in SS(I_-, neg-g))\ (K(p, \langle T_+, T_- \rangle))\}.$$

The boundary sets $S_{s,g}$ and $G_{s,g}$ are defined as follows:

$$S_{s,g} = MAX(VS_{s,g})$$

$$G_{s,g} = MIN(VS_{s,g}).$$

Notice that $S_{0,0}$ and $G_{0,0}$ are equal to the boundary sets S and G defined in section 3.2 of this chapter.

We now prove that any of a range of boundary sets $S_{s,j}$ and $G_{k,g}$ correctly represent $VS_{s,g}$. Notice that theorem 2 is a special case of theorem 5, for which $s=g=j=k=0$.

Theorem 5: Validity of boundary sets representation for $VS_{s,g}$.

Consider the concept learning problem $\langle P, M, \langle I_+, I_- \rangle \rangle$ for which P and M are admissible, and in which I_+ and I_- are non-empty subsets of I.

If $j \geq g$ and $k \geq s$, then

$$(\forall p \in P)((p \in VS_{s,g}) \leftrightarrow (\exists s \in S_{s,j})(\exists g \in G_{k,g})(s \geq p \geq g)).$$

Proof: Consider first the proof that $(\forall p \in P)((p \in VS_{s,g}) \rightarrow (\exists s \in S_{s,j})(\exists g \in G_{k,g})(s \geq p \geq g))$.

Consider arbitrarily chosen

$s \in S_{s,j}$, and $p \in P$ such that $s \geq p$. Then by $s \in VS_{s,j}$,

$$((\exists T_+ \in SS(I_+, pos-s))\ (\forall i \in T_+)\ M(s,i))$$

and by $s \geq p$,

$\{i \in I \mid M(a,i)\} \subseteq \{i \in I \mid M(p,i)\}$.

Thus,

$((\exists T_+ \in SS(I_+, \text{pos-s}))\ (\forall i \in T_+)\ M(p,i))$ (1).

Similarly, for arbitrarily chosen $g \in G_{k,g}$ and $p \in P$ such that $p \geq g$, by $g \in VS_{k,g}$,

$((\exists T_- \in SS(I_-, \text{neg-g}))\ (\forall i \in T_-)\ \neg M(g,i))$.

And by $p \geq g$,

$\{i \in I \mid M(p,i)\} \subseteq \{i \in I \mid M(g,i)\}$

$\{i \in I \mid \neg M(p,i)\} \supseteq \{i \in I \mid \neg M(g,i)\}$.

Thus,

$(\exists T_- \in SS(I_-, \text{neg-g}))\ (\forall i \in T_-)\ \neg M(p,i))$ (2).

By (1) and (2) above, we conclude $p \in VS_{s,g}$.

Next consider the proof that $(\forall p \in P)((p \in VS_{s,g}) \rightarrow (\exists s \in S_{s,j})(\exists g \in G_{k,g})(s \geq p \geq g))$.

We begin by expressing $VS_{s,g}$

$VS_{s,g} = \{p \in P \mid (\exists T_+ \in SS(I_+, \text{pos-s}))\ (\exists T_- \in SS(I_-, \text{neg-g}))\ (K(p, \langle T_+, T_- \rangle))\}$

as the union over all $\langle T_+, T_- \rangle$, of $VS_{0,0}$ associated with that $\langle T_+, T_- \rangle$:

$$VS_{s,g} = \bigcup_{\substack{\langle T_+, T_- \rangle \\ T_+ \in SS(I_+, \text{pos-s}) \\ T_- \in SS(I_-, \text{neg-g})}} VS_{0,0} \text{ associated with } \langle T_+, T_- \rangle$$

Consider, then, arbitrarily chosen $p \in VS_{s,g}$. By theorem 2,

$(\exists s \in S)(\exists g \in G)(s \geq p \geq g)$

where S and G are the boundary sets associated with the $\langle T_+, T_- \rangle$ for which $K(p, \langle T_+, T_- \rangle)$.

And for arbitrary $s \in S$ such that $s \geq p$, either $s \in S_{s,g}$, or there is some p contained in a different version space, $VS_{0,0}'$ associated with

a different $\langle T_+, T_- \rangle'$, for which $p > s$. In this case, again by theorem 2,

$(\exists s' \in S')(s' \geq p \geq s)$

where S' is the set $MAX(VS_{0,0}')$. Now either $s' \in S_{s,g}$, or there is yet another version space containing some more specific pattern. Since I_+ and I_- are assumed to be finite sets, $SS(I_+, \text{pos-s})$ and $SS(I_-, \text{neg-g})$ must also be finite. This sequence of progressively more specific patterns bounding the $VS_{0,0}$ associated with various $\langle T_+, T_- \rangle$ therefore has a maximum element, which by definition belongs to $S_{s,g}$. Therefore,

$(\forall p \in VS_{s,g})\ (\exists s \in S_{s,g})\ (s \geq p)$.

And since for $j \geq g$, $S_{s,g} \subseteq S_{s,j}$, we have

$(\forall p \in VS_{s,g})\ (\exists s \in S_{s,j})\ (s \geq p)$ for $j \geq g$.

A similar argument applies to the set $G_{k,g}$, yielding

$(\forall p \in VS_{s,g})\ (\exists g \in G_{k,g})\ (p \geq g)$ for $k \geq s$.

Therefore, the theorem is proven:

$(\forall p \in VS_{s,g})\ (\exists s \in S_{s,j})\ (\exists g \in G_{k,g})\ (s \geq p \geq g)$ for $j \geq g$, $k \geq s$.

Finally, we prove that algorithm described in chapter 6 for revising the sets $S_{s,g}$ and $G_{s,g}$ in response to a new training instance is correct. Theorems 6 and 7 consider the case is which the new training instance is a positive and negative instance, respectively. Theorem 6 is a generalization of theorem 4, theorem 7 a generalization of theorem 3.

Theorem 6: Revising boundary sets given a positive training instance.

Consider the concept learning problem $CP = \langle P, M, \langle I_+, I_- \rangle \rangle$ with

associated $VS_{s,g}$, $S_{s,g}$, and $G_{s,g}$, and a second concept learning problem $CP' = \langle P, M, \langle I_+', I_- \rangle\rangle$ with associated $VS_{s,g}'$, $S_{s,g}'$, and $G_{s,g}'$, and where $I_+' = I_+ \cup \{i\}$.

The sets $S_{s,g}'$ and $G_{s,g}'$ may be determined from the sets $S_{s,g}$, $S_{s-1,g}$ and $G_{s,g}$ as follows:

$$S_{s,g}' = MAX(S_{s-1,g} \cup MAX(\{p \in VS_{s,g} | M(p,i)\}))$$

$$G_{s,g}' = \{p \in G_{s,g} | (\exists s \in S_{s,g}')(s \geq p)\}.$$

Proof: We begin by proving the first of the two equalities. Assume that the cardinality of I_+ is pos, and the cardinality of I_- is neg.

Then the cardinality of I_+' is pos+1. Consider the version space $VS_{s,g}'$.

$$VS_{s,g}' = \{p \in P | (\exists T_+ \in SS(I_+', pos+1-s))\ (\exists T_- \in SS(I_-, neg-g))\ (K(p, \langle T_+, T_- \rangle))\}.$$

Notice that the set of pos+1-s element subsets of I_+' may be written as the union of two disjoint subsets as follows:

$$SS(I_+', pos+1-s) = A \cup B,$$

where A is the set of those subsets of I_+' that do not contain the new instance i.

$$A = \{X \in SS(I_+', pos+1-s) | \neg(i \in X)\}$$

$$= SS(I_+, pos-(s-1))$$

and B contains those subsets of I_+' which contain the new instance, i.

$$B = \{X \in SS(I_+', pos+1-s) | (i \in X)\}.$$

$VS_{s,g}'$ may therefore be represented as the union of two sets:

$$VS_{s,g}' = \{p \in P | (\exists T_+ \in A)\ (\exists T_- \in SS(I_-, neg-g))\ (K(p, \langle T_+, T_- \rangle))\}$$

$$\cup \{p \in P | (\exists T_+ \in B)\ (\exists T_- \in SS(I_-, neg-g))\ (K(p, \langle T_+, T_- \rangle))\}$$

$$VS_{s,g}' = VS_{s-1,g} \cup \{p \in P | (\exists T_+ \in B)\ (\exists T_- \in SS(I_-, neg-g))\ (K(p, \langle T_+, T_- \rangle))\}$$

Since each element in B contains the new instance i, the second set in the above union may be rewritten as follows:

$$\{p \in P | M(p,i) \wedge (\exists T_+ \in SS(I_+, pos-s))\ (\exists T_- \in SS(I_-, neg-g))\ (K(p, \langle T_+, T_- \rangle))\}$$

$$= \{p \in P | M(p,i) \wedge (p \in VS_{s,g})\}$$

$$= \{p \in VS_{s,g} | M(p,i)\}.$$

Thus,

$$VS_{s,g}' = VS_{s-1,g} \cup \{p \in VS_{s,g} | M(p,i)\}. \qquad (1).$$

But since

$$MAX(X \cup Y) = MAX(MAX(X) \cup MAX(Y)),$$

The first half of the theorem is proven:

$$S_{s,g}' = MAX(VS_{s,g}') = MAX(S_{s-1,g} \cup MAX(\{p \in VS_{s,g} | M(p,i)\}).$$

We now prove that $G_{s,g}' = \{p \in G_{s,g} | (\exists s \in S_{s,g}')(s \geq p)\}$.

Because both $G_{s,g}'$ and $G_{s,g}$ are consistent with any subset of negative instances in SS(I_-, neg-g), they differ only in that patterns contained in $G_{s,g}'$ must be consistent with pos+1-s instances from I_+' whereas patterns contained in $G_{s,g}$ need be consistent with only pos-s instances from I_+. But since

$$I_+' = I_+ \cup \{i\}$$

each element in $G_{s,g}'$ must also be consistent with at least pos-s elements of I_+. Therefore,

$$G_{s,g}' \subseteq G_{s,g}.$$

A pattern p can be consistent with pos+1-s instances from I_+' and neg-g instances from I_- only if it satisfies $(\exists s \in S_{s,g}')(s \geq p)$.

Thus,

$$G_{s,g}' = \{p \in G_{s,g} \mid (\exists s \in S_{s,g}')\ (s \geq p)\}$$

and the theorem is proven.

Theorem 7: Revising boundary sets given a negative training instance.

Consider the concept learning problem $CP = \langle P, M, \langle I_+, I_- \rangle\rangle$ with associated $VS_{s,g}$, $S_{s,g}$, and $G_{s,g}$, and a second concept learning problem $CP' = \langle P, M, \langle I_+, I_-' \rangle\rangle$ with associated $VS_{s,g}'$, $S_{s,g}'$, and $G_{s,g}'$, and where $I_-' = I_- \cup \{i\}$.

The sets $S_{s,g}'$ and $G_{s,g}'$ may be determined from the sets $S_{s,g}$, $G_{s,g}$, and $G_{s,g-1}$ as follows:

$$G_{s,g}' = \text{MIN}(G_{s,g-1} \cup \text{MIN}(\{p \in VS_{s,g} \mid \neg M(p,i)\}))$$

$$S_{s,g}' = \{s \in S_{s,g} \mid (\exists g \in G_{s,g}')(s \geq g)\}.$$

Proof: This theorem is the dual of theorem 6. Its proof is analogous to the proof of that theorem.

3.6 Summary

A class of concept learning problems has been defined, and a formal justification of the version space approach provided. The boundary sets representation for version spaces was proven correct for a range of admissible pattern languages including any countably infinite language (theorem 2). The procedure described in chapter 2 for revising these boundary sets in response to training instances was also proven correct (theorems 3 and 4).

A more general definition of version spaces was provided for dealing with inconsistent concept learning problems. The correctness of the boundary sets representation for this general definition of version spaces was proven (theorem 5). The procedure described in chapter 5 for revising the boundary sets for this general version space was also proven correct (theorems 6 and 7).

Chapter 4

Uses for Version Spaces of Partially Learned Concepts

4.1 Introduction

Because version spaces summarize the range of plausible concept descriptions consistent with a given set of training instances, they have important applications to problems of applying and further refining partially learned concepts. Although the issue of working with and representing partially learned concepts has received little attention, it is an important practical problem to be faced if learning programs are to acquire information in a useful form. In this chapter, we consider the use of version spaces to describe what is known about a given concept when available training data is insufficient to determine a unique concept description. This property of version spaces is useful for tasks such as:

1) Making use of partially learned concepts for recognizing new instances of the concept.

2) Selecting useful training instances to direct future learning.

3) Combining partially learned concepts from different sets of training data.

This chapter discusses the above uses of version spaces which stem from their ability to summarize the information contained in the training instances with respect to the concept description language.

4.2 *Using Incompletely Learned Concepts*

Since a version space contains all concept descriptions consistent with the data, it provides a direct indication of the precision with which a concept has been learned. A concept is "completely" learned when the version space contains only a single concept description: a unique concept description consistent with the training instances. If the version space contains many concept descriptions, the identity of the target concept has been only partially determined, and additional training instances are needed to choose among the current possibilities.

It is unrealistic in many problems to expect sufficient training data to be available to define the target concept completely. In the Meta-DENDRAL problem domain, for instance, it is rare that sufficient chemical data are available to uniquely define the chemical concepts learned there. Many desirable training molecules are difficult or impossible to synthesize, and even for available molecules the cost of obtaining training data is substantial. A more basic problem is that it is possible to define concept description languages for which any finite set of training instances is insufficient to determine a unique concept description.

Any performance program which must *use* an incompletely learned concept to classify possible new instances, must have a means of determining how reliable that classification is. The learning program must therefore provide the partially learned concept to the performance program, along with some indication of its reliability.

One possible strategy for using partially learned concepts is to select one of the plausible concept descriptions from the version space (perhaps using outside

knowledge of the domain), and to attach some numerical "certainty" to this concept description.

This strategy of choosing a single concept description with an associated "certainty" does not take full advantage of the information provided by the version space concerning the identity of the target concept. The following example illustrates this point.

Assume that there are several concept descriptions in the version space, and that one concept description, D, is chosen, and assigned some probability of being the correct description of the target concept. Consider instance 1 which is categorized as a positive instance by every concept description in the version space, and instance 2 which is categorized as a negative instance by every concept description except D. Instance 1 should be given a very high probability of being a positive instance, since its classification does not depend upon whether D is the correct description of the concept - *every* concept description in the version space classifies instance 1 as a positive instance. On the other hand, the classification of instance 2 depends strongly upon whether D is the correct description of the target concept. But any scheme which considers only concept description D cannot be aware of this distinction between instance 1 and instance 2.

An attractive strategy for classifying new instances is to provide the entire version space as the set of alternate plausible descriptions of the partially learned concept. As is shown in the following subsections, the version space may be used to classify certain instances with the same certainty as if the concept were completely

learned. For other instances, it is possible to make a reasonable estimate of the certainty of their classification.

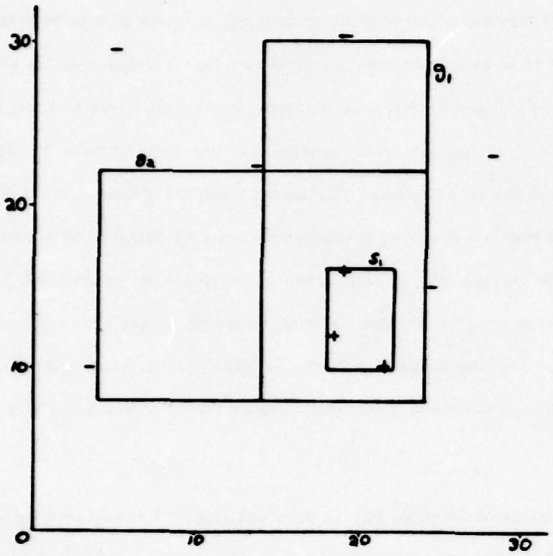
4.2.1 Reliable Classifications Using Partially Learned Concepts

Although for partially learned concepts the exact identity (concept description) of the concept is not known, it *is* known that the correct concept description is one of those contained in the version space. Therefore, instances which are classified in the same way by *all* patterns in the version space may be classified with the same certainty as if the concept description were uniquely determined by the training instances. If this condition is met, then it does not matter which pattern remaining in the version space is the correct description of the target concept - they all yield the same classification.

Consider, for instance, the feature interval learning example in figure 4.1, in which the specific boundary set contains the single pattern s_1 (plotted as a rectangle), and the general boundary set contains the patterns g_1 and g_2. A pattern matches an instance if the corresponding rectangle contains the instance. The version space illustrated in figure 4.1 includes all patterns corresponding to rectangles which both contain s_1, and are contained in either rectangle g_1 or g_2.

Since the version space in figure 4.1 contains many patterns, the associated target concept has not yet been fully determined. Even though much is still to be learned about the identity of the target concept, all instances lying outside both g_1 and g_2 cannot match any patterns in the version space, and therefore should not be

classified as instances of the concept. Alternately, any instance inside the rectangle s_1 matches *every* pattern in the version space. It is therefore an instance of the target concept regardless of which concept description in the version space truly represents the target concept.



G boundary set
g1: 14<x<24, 8<y<30
g2: 4<x<24, 8<y<22

S boundary set
s1: 18≤x≤22, 10≤y≤16

Figure 4.1 Version space of a partially determined concept.

As new training instances are presented, the rectangle s_1 will become larger, while g_1 and g_2 will become smaller. Consequently, a smaller number of patterns will be delimited by these version space boundaries, and, as expected, a larger proportion of all possible instances will be reliably classified by the corresponding version space description of the target concept.

In the above example, it is not necessary to match each pattern in the version space to an instance in order to determine whether (a) all patterns match the instance, (b) all patterns do not match the instance, or (c) some patterns match while others do not. *Regardless of the concept description language, it is necessary only to test the patterns contained in the version space boundary sets, S and G, in order to determine which of these three conditions is met.*

If every pattern in the boundary set S matches an instance, then all patterns in the version space match that instance. This is true because all patterns in the version space are more general than some pattern in S. Similarly, if no pattern in the boundary set G matches an instance, then no pattern in the version space matches that instance (all patterns in the version space are more specific than some pattern in G). Thus, the following rules summarize the method for obtaining a reliable classification of a potential instance of the target concept.

```
IF all patterns in S match this instance,
        THEN this is an instance of the concept
ELSE IF no pattern in G matches the instance,
        THEN this is not an instance of the concept
ELSE there are alternate possible interpretations of the training
     data (alternate patterns in the version space) which yield
     alternate classifications of this instance.
```

It should be kept in mind that in referring to "reliably" classified instances, we mean *reliable with respect to the given concept description language, and the presented training instances.* If the chosen concept description language does not contain a correct description of the concept described by the training instances, then of course any method of classifying new instances cannot be reliable. It is possible to deal with inconsistent data in a reasonable manner using the extended method described in chapter 5. The above strategy for "reliably" classifying instances then becomes a method for classifying instances based upon assumptions concerning the maximum number of undetected errors in the observed training instances.

One additional note on efficiency. It is possible to improve upon the efficiency of the above strategy when the boundary set G contains many patterns. In such cases, it may be more efficient to determine a single pattern which is more general than all the patterns in G and to first test the instance against this pattern. If the instance does not match this pattern, it cannot match any of the patterns contained in

G, or any pattern in the version space. If the instance does match this pattern, it must still be tested against the patterns in G and S. In the Meta-DENDRAL implementation of this classification procedure using version spaces, this efficiency measure is used to quickly pretest instances. Because the boundary sets in that program often contain dozens of patterns, this strategy improves the overall efficiency of the testing procedure.

4.2.2 Estimating Certainty of Unreliable Classifications

For instances which cannot be reliably classified as described above, it may be *useful to estimate the classification of the instance.* In such cases, where the training data is insufficient to determine a reliable classification, additional knowledge about the problem domain should play an important role. It may be possible using such knowledge to reject certain concept descriptions, or to rank by plausibility the alternate concept descriptions. The use of domain specific knowledge in this manner is an attractive area for future work, and has not been considered except in a few obvious ways in the Meta-DENDRAL implementation of the version space approach. Here we present one strategy in which it is assumed that all patterns in the version space are equally plausible as correct descriptions of the target concept.

If all patterns in the version space are considered equally plausible descriptions of the target concept, then it is reasonable to classify the new instance by determining what proportion of these patterns classify the instance as a positive instance. The instance may be classified according to this proportion, and the proportion taken as the certainty of the classification.

Unfortunately, determining exactly how many patterns in a given version space do and do not match an instance is not always simple. One method is to assume the instance in question is a positive instance, and update the version space accordingly. The number of patterns contained in the resulting version space is the number of patterns in the original version space which match the new instance. The ratio of this number to the number of patterns in the original version space yields the proportion of patterns in the original version space which match the instance.

For the feature interval learning problem, it is fairly easy to work out this strategy. In order to simplify the problem, we assume that the language of feature intervals allows only integers (i.e., $4<x<7$ is allowed, but $4.5<x<7.5$ is not). Then to count the number of patterns in the version space of figure 4.1, we simply count those rectangles contained in either g_1 or g_2 which contain the rectangle s_1. Since each rectangle represents the boundaries of a pair of intervals, and since each interval involves two relations, each of which may be either $<$ or $\leq$, the number of patterns in the version space is approximately[1] 2^4 times the number of counted rectangles. We calculate the number, N, of patterns in the version space of figure 4.1 as follows:

$N \approx 16 \cdot (A+B-C)$

A = number of rectangles contained in g_1, which contain s_1

$= |(14-18)| \cdot |(22-24)| \cdot |(8-10)| \cdot |(16-30)|$

$= 5 \cdot 3 \cdot 3 \cdot 15 = 675$

[1] For convenience, we ignore the small effect on the total count of patterns arising from the fact that each maximally general pattern may contain *only* the $<$ relation.

B = number of rectangles contained in g_2, which contain s_1

$= 15 \cdot 3 \cdot 3 \cdot 7 = 945$

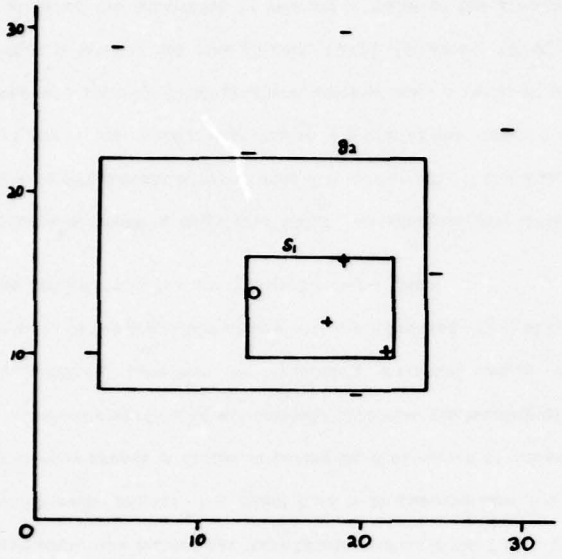
C = number of rectangles contained in both g_1 and g_2, which contain s_1

$= 5 \cdot 3 \cdot 3 \cdot 7 = 315$

$N \approx 20{,}880$.

There are approximately 20,880 patterns in the current version space. Now suppose the instance (13, 14) is presented to be classified as either a positive or negative instance of the target concept. In order to determine the number of patterns in the version space which match this instance, we update the version space *under the assumption that the new instance is a positive instance.* Figure 4.2 illustrates the resulting version space, in which the new instance is represented by a circle[1].

[1] Notice that in figure 4.2, the pattern g_1 has been removed from the boundary set G. As explained in chapter 2, this is because g_1 does not match the new "positive" instance.



G boundary set
g2: 4<x<24, 8<y<22

S boundary set
s1: 13≤x≤22, 10≤y≤16

Figure 4.2 Effect on version space of treating (13, 14) as a positive training instance.

In the resulting version space, there are approximately 10,080 patterns. The probability that the new instance is a positive instance is therefore ≈ 10080/20880 ≈ 0.48, under the assumption that all patterns in the version space are equally likely descriptions of the target concept.

The calculation of the number of patterns in a version space, and hence an estimate of the reliability of a classification based upon a partially learned concept, is straightforward for the feature interval learning problem. For more complex structural description languages, the calculations are more involved, but the general approach is the same. The assumption that all concept descriptions in the version space are equally likely descriptions of the target concept does not take into account information concerning the relative probabilities of the alternate concept descriptions. Still, the probability of classification based on this assumption provides a reasonable estimate based upon the information available.

The above section describes the use of version spaces for summarizing and applying partially learned concepts. With respect to the chosen concept description language, it is possible to determine how precisely the training data has described the identity of the target concept. As described above, it is also possible to classify subsequent instances on the basis of an incomplete knowledge of the identity of the target concept. The following section considers the related problem of generating informative training instances to direct future refinements to partially learned concepts.

4.3 Requesting New Training Instances

This section discusses requesting new training instances to direct further refinement of an incompletely learned concept. The issue of choosing good *sequences of training instances* has been addressed by many researchers [Bruner, 1956], [Popplestone, 1969], [Simon, 1973], [Buchanan, 1974], [Smith, 1977], but an understanding of general methods for intelligent training instance generation has yet to be developed. The problem has many facets, and could itself form the basis for a substantial research project. The intent here is to describe how the information contained in version spaces is crucial to generating a sequence of instances whose (externally provided) classification will optimally determine the identity of the target concept. In addition, guidelines for requesting instances which control the intermediate sizes of the S and G boundary sets are discussed. Both of these issues are illustrated using a simple feature interval learning problem.

The problem of choosing informative training instances is very different if the identity of the target concept is known than if it is unknown. If the agent choosing the instances knows the concept, the problem is similar to that faced by intelligent computer assisted instruction (ICAI) programs, which present a series of examples to a student. Recent work in this area [Brown, 1975], [Sleeman, 1979], [Clancey, 1979] has stressed the importance of inferring a model of the student's current understanding of a concept as a basis for selecting the next training example.

In contrast, we consider here the problem in which *the learner must choose the next instance*. Thus an instance is chosen whose classification is expected to provide

additional information concerning the target concept. Although instances are chosen without exact knowledge of the target concept, version spaces provide a useful summary of what has been learned so far about the target concept. Thus, in contrast to the problem faced by ICAI programs, an exact model of the "student's" current understanding of the concept is available.

4.3.1 Choosing Instances to Efficiently Determine the Concept

Since concept learning using version spaces corresponds to reducing the number of alternate concept descriptions in the version space, the goal of efficiently learning the target concept may be rephrased as the goal of rapidly reducing the number of concept descriptions in the version space. Intelligent instance generation then becomes quite similar to a game of twenty questions, in which the task is to determine which of a large set of possible hypotheses (concept descriptions in the current version space) is the correct one by posing questions (training instances) which can be answered either yes or no (positive or negative classification). Assuming that all hypotheses are equally likely, the optimal strategy is to ask questions which will rule out *one half or the other of the current hypotheses, depending upon the answer.* The optimal strategy for generating instances (without knowledge of the exact identity of the target concept) is to generate at each step an instance which matches half the concept descriptions in the current version space.

Consider, as an example, the version space for the feature interval learning problem illustrated in figure 4.1. If an optimal strategy for generating training instances without knowledge of the target concept is used, then the expected

number of training instances required to complete learning of the target concept is $\log_2 N$, where N is the number of concept descriptions in the current version space[1]. For the version space of figure 4.1, N=20880. Therefore, we expect that $\log_2 20880 \approx 16$ additional well chosen training instances will be sufficient to learn the target concept associated with that problem. It is interesting to note that given knowledge of the identity of the target concept, it is possible to choose a sequence of 6 training instances to unambiguously describe any feature interval concept[2]. As expected, knowing the target concept allows designing more efficient training sequences.

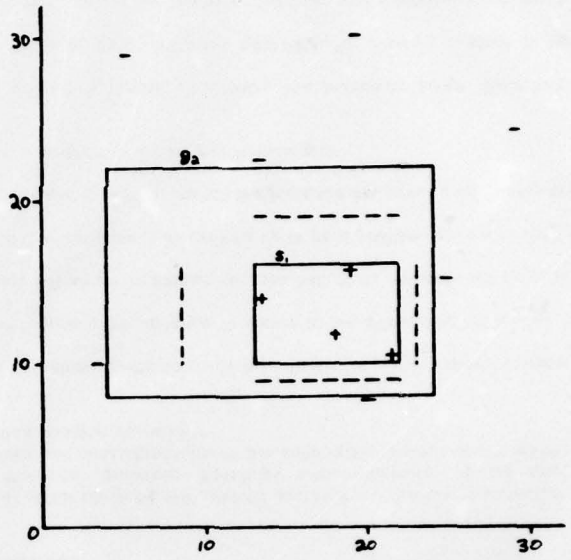
Which instance should be chosen next for the version space in figure 4.1? As shown in the previous section, the instance (13, 14) matches 48% of the patterns in this version space. This instance would therefore make a near optimal choice as the next training instance for updating the version space. Figure 4.2 illustrates the version space which results if the instance (13, 14) is generated, found to be a positive instance, and is then used to update the version space. It comes as no surprise that the most informative new training instances are the instances which are least reliably classified by the version space.

Although it is obvious in retrospect that (13, 14) was an excellent choice for the next training instance, how can such instances be derived in general from the boundary sets representing the current version space? The solution depends upon understanding the combinatorics of the particular pattern language. For the version

[1] The corresponding statement for the game of twenty questions is a well-known result from information theory.

[2] Given the rectangle corresponding to the target concept choose negative training instances at each corner of the rectangle, and positive instances in any two opposing corners.

space illustrated in figure 4.2, the combinatorics may be worked out quite easily. Figure 4.3 shows the version space from figure 4.2, with a dotted line indicating the locus of instances which match half the patterns in the version space, and which are therefore optimal next training instances.



G boundary set
g2: 4<x<24, 8<y<22

S boundary set
s1: 13≤x≤22, 10≤y≤16

Figure 4.3 Optimally informative next training instances (dashed line).

Although it is possible to derive the set of optimal next training instances for the above version space, such an analysis for structural concept description languages is more complex, and exact solutions may be costly in terms of processing time. A detailed analysis of the combinatorics of the individual concept description language is necessary to determine a general rule for generating such optimal instances for learning problems using that language. Of course an approximate procedure for finding instances which match *nearly* half the patterns in the version space would be nearly as useful an asset as an assured optimal procedure. Such approximate methods might be evaluated by comparing their performance against the theoretical optimal $\log_2 N$ performance.

4.3.2 Choosing Instances to Control Boundary Set Sizes

The computational efficiency of the version space approach is closely tied to the sizes of the boundary sets used to represent version spaces. Although the boundary sets are an efficient representation for version spaces when compared with the alternative of listing all patterns in the version space, in many situations the size of these boundary sets is still the limiting factor in applying the approach. One important aspect of the capability to generate new training instances, therefore, is the option of generating training instances to control the sizes of these boundary sets[1].

Generating instances to control the sizes of the version space boundaries is complicated by the fact that it is not known beforehand whether the generated

[1] I am indebted to an anonymous IJCAI referee for suggesting this possibility.

instance will be classified as a positive or negative instance of the target concept. The following rules for generating instances appear to be reasonable even in the absence of this information.

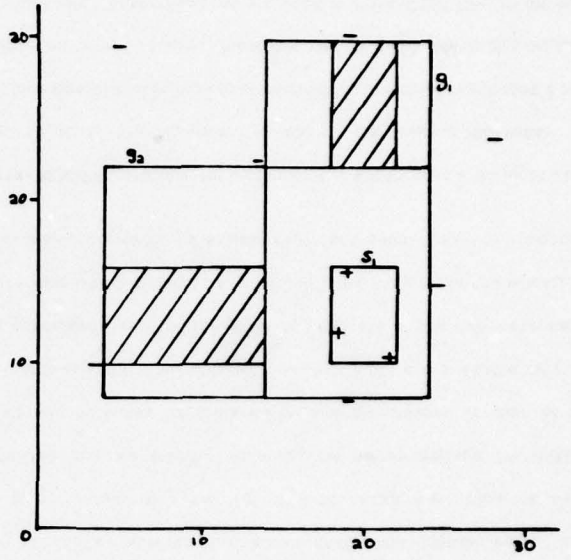
> 1) Generate instances which match half the patterns in each boundary set.
>
> 2) After choosing the specific patterns which are not to match the instance, generate instances whose feature values and relations are consistent with all the constraints except one in each of these specific patterns.

The first rule above tends to shrink boundary sets which already contain multiple patterns. If the generated instance is found to be a positive instance, then the general boundary will shrink to contain only the half of its patterns which match match the instance. If the instance is instead found to be a positive instance, then only the half of the patterns which did not match the instance can lead to additional branching. The effect on the specific boundary set is analogous.

The second rule attempts to minimize the branching which does occur. The strategy is to focus on determining the constraint for a single feature or relation in the pattern, choosing values for the other features and relations which are known to be consistent with the target concept. In other words, the generated instance would match all patterns in the version space if it were not for the single different feature value. When the classification of this new instance is obtained, the only branching in the boundary sets is branching associated with this feature. If, on the other hand, an instance were generated with several feature values which conflict with the specific patterns, then branches of the partial ordering corresponding to altering each

corresponding constraint (as well as combinations of these) would have to be considered.

Briefly consider the above strategy applied to the feature interval learning problem, in which two features, x and y, are to be constrained by patterns, and no relations are involved. Since in this problem the specific boundary can never contain more than a single pattern, there is no danger of branching of this boundary. The general boundary can, however, contain multiple patterns, as it does in figure 4.4. This figure shows the version space illustrated in figure 4.2, with the shaded region indicating instances which satisfy the above rules for generating new instances. These shaded instances either match the specific constraint on allowed values of x, but do not match the constraint on y, or vice versa. Any instance in this shaded area, regardless of its classification, will lead to the same size or smaller boundary sets. Notice that this shaded region includes the instance (13, 14), shown earlier to be a near optimal training instance for reducing the uncertainty of the partially learned concept description.



G boundary set
g1: 14<x<24, 8<y<30
g2: 4<x<24, 8<y<22

S boundary set
s1: 18≤x≤22, 18≤y≤16

Figure 4.4 Training instances which control boundary set sizes (shaded area).

The above strategy always succeeds in reducing or, at worst, not increasing the boundary set sizes for the feature interval learning problem. For structural pattern languages, the situation is complicated by the fact that multiple possible mappings of pattern nodes to instance nodes must be considered. As a result, a guarantee such as the one above is not possible for structural pattern languages. Nevertheless, this appears to be a good strategy for the reasons given above, for controlling the sizes of the version space boundaries for structural languages.

It should be stressed that the above strategy for generating new training instances has not been implemented. The discussion in this section is, as mentioned earlier, preliminary. However, given the demonstrated potential of using version spaces for generating new instances, and the importance of finding short sequences of training instances which completely determine the target concept while keeping the branching of the version space boundaries under control, this appears to be a rewarding area for further research.

4.4 Combining Separately Obtained Results and Merging Concept Descriptions

Yet another use for version spaces is in merging concept descriptions learned from different sets of training data. In particular, suppose that the version space VS_1 is formed from the set of training data T_1, and that the version space VS_2 is formed from a second set of training data, T_2. Then the version space, VS, consistent with the *union* of the training sets T_1 and T_2 is the *intersection* of the version spaces VS_1 and VS_2. Stated another way, a concept description is consistent with all instances in $(T_1 \cup T_2)$ (i.e., it belongs to VS) if and only if it is consistent with T_1 (belongs to VS_1) and is consistent with T_2 (belongs to T_2).

This property leads to a simple method for comparing and combining results learned from separate sets of data. An algorithm is described below which computes the boundary sets representing the intersection of VS_1 and VS_2 directly from the boundary sets representing VS_1 and VS_2.

One could imagine a program learning two concepts, but being uncertain whether these two could instead be expressed as a single concept in the provided language. This may often be the case when it is known that a disjunctive set of concept descriptions is to be learned by a program. In such cases, each disjunct will cover some subset of the positive training instances. A simple method for determining whether any two disjuncts may be replaced by a single disjunct is then of obvious value. If the version spaces associated with each disjunct are available (that is, the version spaces associated with the corresponding subsets of positive instances) then it is easy to determine which disjuncts may be combined. Any disjuncts whose version spaces intersect may be replaced by any concept description in that intersection.

In the Meta-DENDRAL problem, for instance, it is known that a disjunctive set of rules is needed to cover any significant portion of the training instances. Meta-DENDRAL therefore searches for individual rules which cover a significant portion of the positive instances. Almost always, there are several rules which can be replaced by a single stronger rule. Version spaces are used in Meta-DENDRAL to detect such occurrences while, at the same time, computing the merger of the disjuncts.

A second application of the capability to combine results learned from separate sets of training instances lies in the prospect of parallel processing. A convenient

method for taking advantage of multiple processors for concept learning problems is to break the set of available training instances for a given concept into several subsets. Each processor may then compute the version space consistent with one of these subsets. The intersection of the version spaces determined by the set of processors yields exactly the result which would be obtained by a single processor working with the entire set of training instances.

4.4.1 Algorithm for Determining Intersection of Version Spaces

Consider two version spaces, VS_1 and VS_2, represented respectively by the pairs of boundary sets $\langle G_1, S_1 \rangle$, and $\langle G_2\ S_2 \rangle$. The version space, VS, which is the intersection of VS_1 and VS_2 may be represented by the boundary sets $\langle G, S \rangle$, where

$$S = MAX(VS)$$
$$= MAX(VS_1 \cup VS_2)$$
$$= MAX(\{p \in P | (\exists s_1 \in S_1)(\exists s_2 \in S_2)(\exists g_1 \in G_1)(\exists g_2 \in G_2)\ (s_1 \geq p) \wedge (s_2 \geq p) \wedge (p \geq g_1) \wedge (p \geq g_2)\})$$
$$= \{x \in ML(S_1, S_2) |\ (\exists g_1 \in G_1)(\exists g_2 \in G_2)\ (x \geq g_1) \wedge (x \geq g_2)\ \}$$

and

$$G = MIN(VS)$$
$$= MIN(VS_1 \cup VS_2)$$
$$= MIN(\{p \in P | (\exists s_1 \in S_1)(\exists s_2 \in S_2)(\exists g_1 \in G_1)(\exists g_2 \in G_2)\ (s_1 \geq p) \wedge (s_2 \geq p) \wedge (p \geq g_1) \wedge (p \geq g_2)\})$$
$$= \{x \in MU(G_1, G_2) |\ (\exists s_1 \in S_1)(\exists s_2 \in S_2)\ (s_1 \geq x) \wedge (s_2 \geq x)\ \}$$

where

$$ML(S_1, S_2) = MAX(\{p \in P |\ (\exists s_1 \in S_1)(\exists s_2 \in S_2)\ (s_1 \geq p) \wedge (s_2 \geq p)\})$$

$$MU(G_1, G_2) = MIN(\{p \in P |\ (\exists g_1 \in G_1)(\exists g_2 \in G_2)\ (p \geq g_1) \wedge (p \geq g_2)\})$$

The last expression for each set is written in a way which suggests an algorithm for finding S and G from the sets S_1, S_2, G_1, and G_2. In particular, S may be computed by finding the subset of patterns from $ML(S_1, S_2)$ which are more specific than or equal to some pattern in both G_1 and G_2. The set G may be computed in an analogous manner.

The functions ML and MU have been implemented for the Meta-DENDRAL language of chemical concept descriptions, and are used there to compute the intersection of version spaces formed from different clusters of positive instances. When the version spaces associated with two different clusters of positive instances are found to intersect, they are replaced by their intersection. This version space contains all concept descriptions consistent with the union of the two sets of positive instances.

4.5 Summary

The boundary sets G and S determined from a given set of training instances provide a useful summary of those instances with respect to the chosen concept description language. Several important uses of this information for problems related to concept learning were considered. In particular, we have discussed the uses of version spaces to classify possible instances of the target concept even when the concept has been only partially learned, to generate useful new training instances to direct future learning while controlling the sizes of the version space boundary sets, and to combine separately learned concept descriptions.

Chapter 5

Learning in Less Perfect Situations

5.1 Introduction

The candidate elimination algorithm as presented in chapters 2 and 3 finds all concept descriptions within a predetermined language which are consistent with *every* observed training instance. In problems for which such concept descriptions exist, this approach provides the solution to the concept learning problem. In other problems, however, there may not be any concept description consistent with each training instance. This chapter presents an extension to the version space approach which allows dealing with this later class of concept learning problems.

When no concept description is consistent with all training instances, the problem may be caused by either incorrect training instances, a deficiency in the concept description language, or when several disjoint concept descriptions are needed to cover the entire class of training instances. Although these three causes appear to be distinct, they lead to a common problem for the concept formation algorithm. Therefore, these will be treated as different causes of the single problem of inconsistency of training instances with respect to the concept description language - for short, the *inconsistency problem*.

Many artificial intelligence approaches to machine learning have not addressed the inconsistency problem. Statistical approaches of pattern recognition and adaptive

control have, on the other hand, emphasized its importance. Artificial intelligence approaches to learning have been able to ignore the inconsistency problem largely because they have been developed in domains with reliable data. As we learn enough about learning to move away from simple domains, inconsistency problems will become increasingly common. Few measurements taken in the real world are completely noise-free, and it is unlikely that our data representations will be able to model the real world so faithfully that the distinctions between similar situations will always be representable. Thus, for learning techniques to be applicable to a large number of new problems, we will have to face squarely the problem of inconsistency.

In the rest of this chapter, several sources of inconsistency are discussed. An extension of the version space approach is presented which allows learning concepts in the presence of inconsistency. This extended approach determines concept descriptions consistent with the largest possible subsets of training instances for concept learning problems involving only a few inconsistencies. A less costly heuristic method is also presented which attempts to determine optimal concept descriptions according to a scoring function based upon numbers of positive and negative instances with which the concept description is consistent.

5.2 Three Inconsistency Problems

An *inconsistent* concept learning problem refers here to any concept learning problem for which there is no perfect solution; that is, any problem for which no concept description may be found which is consistent with every training instance. As noted above, there are several possible causes of such inconsistency. Here we

briefly describe three possible causes in order to illustrate the range of inconsistent concept learning problems.

5.2.1 Incorrect Training Instances

One cause of inconsistency in concept learning problems is incorrect training instances. Incorrect training instances do not accurately represent the target concept for some reason, and may or may not result in inconsistency as defined above. Two causes of incorrect training instances are apparent in the Meta-DENDRAL problem. Meta-DENDRAL learns production rules that predict which bonds in a given molecule will break within a mass spectrometer. For this program, rules are formed from training instances consisting of molecular bonds which break or do not break inside the mass spectrometer.

Unfortunately, such training instances are not directly available to Meta-DENDRAL. Instead of bonds which break and do not break, the available training data consist of masses of the resulting molecular fragments. In general, an observed fragment mass may be attributable to any of several potential broken bonds within the molecule. Thus, the available training data in this case are not the needed training instances, but rather data from which the training instances must themselves be inferred. The ambiguity of mapping the available data of observed molecular fragments into inferred training instances of broken and unbroken bonds results in a set of partly incorrect, and often inconsistent training instances.

In addition to the ambiguity in inferring training instances in the desired form

from available data, Meta-DENDRAL must deal with a second source of training instance error: the available data themselves may be incorrect. The available data are obtained from the mass spectra of known molecules. The resulting spectra may contain spurious peaks resulting from impurities in the input sample. At the same time, fragmentations which do occur may not always be detected by the instrument. Thus, incorrect positive instances as well as incorrect negative instances are implied by the observed data.

Whatever the source of incorrect data, it is useful to divide incorrect training instances into two classes. Training instances incorrectly presented as positive instances will be described as *false positive instances*, while instances incorrectly presented as negative instances will be called *false negative instances*. This dichotomy of data errors has important implications for detecting and dealing with inconsistency.

5.2.2 Insufficient Concept Description Language

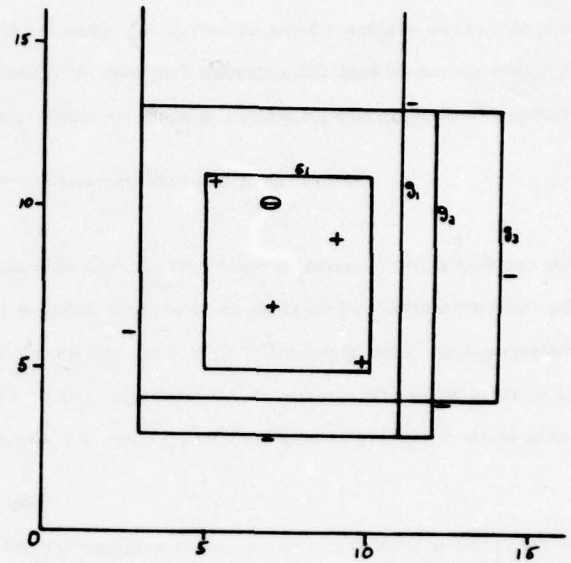
A second possible cause of inconsistent concept learning problems is that the language chosen for describing concepts simply does not contain a correct description of the target concept. The problem of finding a language rich enough to represent all possible concepts in a given domain is a difficult one. In the Meta-DENDRAL program, for instance, it is known that the language of chemical substructures used to form rules is not sufficient to express all possibly meaningful rules of mass spectroscopy. But chemists themselves do not know all the important features needed for a better language, and no one yet knows how to write a program which successfully designs its own concept description language. It appears, therefore, that this cause of inconsistency is unavoidable in at least some concept learning problems.

5.2.3 Disjunctive Concepts

Inconsistency in concept learning problems may also arise from the fact that training instances presented as belonging to a single target concept may have to be represented as a disjunction of concept descriptions from the chosen language. Although this might be interpreted as a case in which an insufficient concept description language is being used (a sufficient language is the language of arbitrary disjunctions of patterns in the current language), it is a special case worth singling out. For in this case, it may be possible to successfully cover all training instances by learning a disjunctive set of concept descriptions which taken together are sufficient to describe the target concept. For example, in the Meta-DENDRAL program, a *set* of rules is typically required to explain a significant amount of the training data.

5.3 Detecting Inconsistency with Version Spaces

The first step toward dealing with inconsistency in a concept learning problem is detecting the fact that an inconsistency exists. When an inconsistency arises, the version space becomes empty. In the basic version space approach the boundary sets S and G collapse into empty sets, giving an immediate indication that there are no concept descriptions consistent with all the training instances. This may happen either when a new negative training instance matches every pattern contained in the S version space boundary, or when a new positive instance matches no pattern contained in G.



G boundary set
g1: 3<x<11, 3<y<∞
g2: 3<x<14, 4<y<13
g3: 3<x<12, 3<y<13

S boundary set
s1: 5≤x≤10, 5≤y≤11

Figure 5.1 An inconsistent interval learning problem. New training instance (circled) yields an inconsistent instance set.

Consider again the class of learning problems described in chapter 2 in which concepts are represented as feature value intervals. Figure 5.1 shows a consistent set of positive and negative training instances (uncircled + and - points), and the patterns contained in the associated version space boundary sets. Recall that in this class of problems, concept descriptions correspond to rectangles which match the instances contained within them. If the circled negative instance, (7, 10), is presented as the next training instance, an inconsistency arises. No legal concept description (rectangle) is consistent with this new instance as well as all previous training instances. The candidate elimination algorithm detects this fact by noting that the pattern in the S boundary matches this negative instance. Since all patterns in the version space are more general than the S boundary, they all match the instance.

Once an inconsistency is found to exist, the concept learning problem becomes a problem of finding a concept description which is in some way a "best fit" to the observed data. That is, finding a concept description which is consistent with some reasonable subset of the training instances. Detecting an inconsistency does not, of course, indicate the cause of the inconsistency. In the above example it is impossible to know whether the new positive instance is incorrect (a "stray"), whether some previous instance was incorrect, or whether some cause other than data error is responsible for the detected inconsistency.

5.4 Learning and Inconsistency

A reasonable approach to dealing with inconsistency is to try to find the largest

consistent subsets of training instances, and to find concept descriptions consistent with these subsets. This is an imposing task, since testing n-element subsets of the training instances is a combinatorially explosive problem. Nevertheless, the goal of finding these so-called maximally consistent concept descriptions is a useful one to keep in mind when considering how to deal with inconsistent learning problems.

Maximally consistent concept description: A concept description is maximally consistent with a set of positive and negative training instances if there is no concept description consistent with a larger number of training instances.

The following sections describe an extension of the version space approach which allows *learning* in the presence of inconsistency. In this extended approach, the term version space is generalized to refer to a set of concept descriptions consistent with any of a range of possible training instances, rather than a single, definite set of training instances. Several such version spaces, consistent with progressively smaller classes of training instances are maintained by the program. Several strategies are discussed which utilize these multiple version spaces for learning in the presence of inconsistency. An optimal solution, which determines maximally consistent concept descriptions, is presented for problems involving a small number of inconsistencies. A heuristic approach is presented to deal more efficiently with more severe inconsistency.

When an inconsistency is detected by the collapse of the version space consistent with all training instances, the program shifts its attention to the version space consistent with the next largest consistent subset of training instances, and constructs new version spaces consistent with yet smaller subsets of the training

instances, as needed. Although this is a backtracking style approach, it differs from many backtracking approaches in two important respects:

1) Training instances need not be stored or reexamined. Instead, the information needed for backtracking is summarized in the additional boundary sets - sets whose size remains approximately constant with the number of observed training instances.

2) These additional sets delimit concept descriptions consistent with many different subsets of the training instances without explicitly examining each such subset.

5.4.1 Multiple Version Spaces

The extended version space approach operates on sets of concept descriptions in much the same way that the basic version space approach operates on the boundary sets G and S. Given a set of p positive and n negative training instances, the following boundary sets of concept descriptions are used to delimit the version spaces of various subsets of the training instances.

$S_{s,j}$: The set of maximally specific concept descriptions consistent with any subset of p-s positive training instances, and n-j negative instances.

$G_{k,g}$: The set of maximally general concept descriptions consistent with any subset of n-g negative training instances, and p-k positive instances.

Notice that $S_{0,0}$ and $G_{0,0}$ are the same as the boundary sets S and G used in the basic version space approach. The boundary sets defined above delimit several version spaces, defined as follows:

$VS_{s,g}$: The set of concept descriptions consistent with *any* subset of p-s positive instances and n-g negative instances.

$VS_{s,g}$ can be represented by the boundary sets $S_{s,j}$ and $G_{k,g}$, where $j \geq g$ and $k \geq s$ (this is proven in Theorem 6 of chapter 3). That is, the concept descriptions contained in $VS_{s,g}$ are exactly those which lie between $S_{s,j}$ and $G_{k,g}$ in the general to specific ordering. More formally,

For any j and k such that $j \geq g$ and $k \geq s$,

$$VS_{s,g} = \{p \in P \mid (\exists x \in S_{s,j})(\exists y \in G_{k,g})\ (x \geq p \geq y)\}$$

where P is the set of all concept descriptions.[1]

In order to simplify notation, and since we will only consider pairs of boundary sets $S_{s,j}$ and $G_{k,g}$ for which $j \geq g$ and $k \geq s$, we will often drop the subscripts j and k, and refer simply to S_s and G_g. The exact values of j and k do not affect the use of $S_{s,j}$ and $G_{k,g}$ in representing $VS_{s,g}$, as long as $j \geq g$ and $k \geq s$.

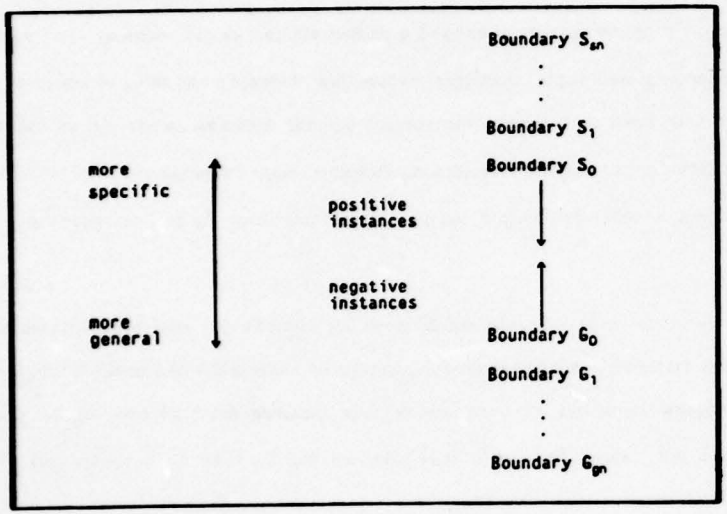
5.4.2 Learning Using Multiple Version Spaces

In the extended version space approach, as in the basic approach, learning occurs when candidate concept descriptions are eliminated from the version space. All boundary sets S_s and G_g are revised in response to each training instance so that at each step, $VS_{s,g}$ is correctly represented.

In order to deal with inconsistent concept learning problems, two series of

[1] Note the relation $\geq$ is used above in two different senses. When applied to integers it refers to the usual "greater than or equal" relation. When applied to concept descriptions, it refers to the "more specific than or equal" relation.

boundary sets S_0, S_1, ... S_{sn}, and G_0, G_1, ... G_{gn} are maintained as depicted in figure 5.2. Each pair of general and specific boundary sets represents a different version space corresponding to different subsets of the training instances. As before, training instances force the S_s boundaries to become more general and the G_g boundaries to become more specific. An inconsistency is detected when S_0 becomes more general than G_0 (the boundaries cross), indicating that the version space $VS_{0,0}$ is empty. When this occurs, the program falls back to the version spaces $VS_{1,0}$ and $VS_{0,1}$ which at that point contain all concept descriptions maximally consistent with the training instances.



Multiple Version Space Boundaries
Figure 5.2

Concept descriptions maximally consistent with the training data can always be determined using this approach *provided* there is a consistent subset of at least p-sn positive instances, and n-gn negative instances in the training data. If this condition is not met, every G_g boundary will become more specific than every S_s boundary, indicating that the version spaces $VS_{0,0}$ through $VS_{sn,gn}$ all are empty.

When the above approach falls, a heuristic approach may be used. In the heuristic approach, the program selects at each step a version space consistent with the largest possible number of training instances. On the basis of this choice, it decides which additional boundary sets to retain, which ones to drop from consideration, and which sets should be added to the list it is keeping[1]. Although this heuristic approach attempts to find maximally consistent concept descriptions, it cannot be assured of finding optimal solutions to inconsistent learning problems. This is due to the fact that (1) the program selects a set of "current best version space boundaries", rather than all possible boundaries, and (2) when new boundaries sets are added, they are approximated (as described below) so that past training instances need not be reexamined.

Before considering alternate strategies for learning using multiple version spaces, we examine the algorithm for updating the boundary sets. The boundary sets must be initialized using an initial set of positive and negative training instances. Each set is then updated in response to subsequent training instances in a manner which is independent of the order in which training instances are presented.

[1] As discussed below, when new boundary sets are added in the middle of processing a sequence of training instances, only an approximation to the true boundary set is available unless previous training instances are reconsidered.

5.4.2.1 The Algorithm

The algorithm for updating the boundary sets S_s and G_g utilizes the same operators for modifying the boundary sets as are used by the basic approach described in chapter 2, and involves an additional comparison between boundary sets. In order to understand the algorithm for updating the boundary sets, consider an example.

Suppose the set S_s, consistent with p-s of the p observed positive training instances, has been initialized. When a new positive instance is observed, S_s must be updated to the set of maximally specific concept descriptions consistent with *any* p+1-s of the p+1 positive instances. Any concept description in the new S_s must be consistent with either (1) s+1 of the original p positive instances, or (2) s of the original p positive instances *and* the new instance. The updated set S_s is therefore obtained by taking the maximally specific elements in the union of the following two sets: (1) the old boundary set S_{s-1}, and (2) the set S_s updated as in the basic approach to be consistent with the new positive instance. The first of these sets bounds the concept descriptions consistent with p+1-s of the first p positive instances, while the second set bounds those descriptions consistent with s of the first p instances as well as the new instance. In this way the set S_s may be updated in an incremental fashion, *without explicitly considering all subsets of p+1-s of the p+1 instances.*

The boundary set S_s may be initialized by examining the first s+1 positive instances, and setting S_s to the set of maximally specific concept descriptions

consistent with any of these instances. Similarly, G_g may be initialized by examining the first g+1 negative instances, and finding the set of maximally general concept descriptions consistent with any of these. This is a generalization of the procedure for initializing S and G in the basic version space approach.

Once the boundary sets S_0 through S_{sn}, and G_0 through G_{gn} have been initialized, they are updated with each new training instance as described below. The correctness of this algorithm is proven in theorems 6 and 7 of chapter 3. The procedures UPDATE-G(G, S, i) and UPDATE-S(S, G, i) refer to the procedures described in chapter 2, used there to update the sets S and G. They yield the set of minimally more specific (general) concept descriptions consistent with the new instance.

Algorithm for Updating Multiple Version Space Boundaries

IF the new instance, I, is a negative instance

THEN BEGIN

FOR g=gn to 0 DO

G_g ← MIN(G_{g-1} ∪ UPDATE-G(G_g, S_{sn}, i));

FOR s=sn to 0 DO

S_s ← the subset of patterns of S_s which are more specific than some pattern in G_{gn};

END

ELSE IF the new instance is a positive instance

THEN BEGIN

FOR s=sn to 0 DO

S_s ← MAX(S_{s-1} ∪ UPDATE-S(S_s, G_{gn}, i));

FOR g=gn to 0 DO

G_g ← the subset of patterns of G_g which are more general than *some pattern in* S_{sn};

END;

Where MAX(X) and MIN(X) are the sets of maximally specific and maximally general patterns in the set X:

$MAX(X) = \{x \in X \mid (\forall z \in X)\ \neg(z \geq x)\}$

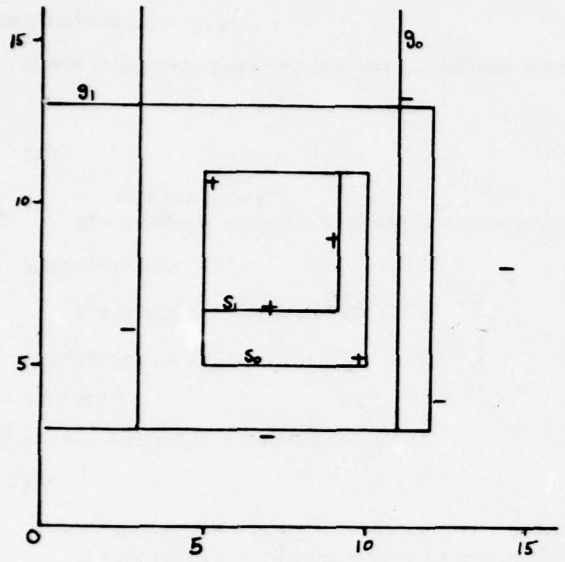
$MIN(X) = \{x \in X \mid (\forall z \in X)\ \neg(x \geq z)\}$

Notice that each set S_s is pruned to include only patterns more specific than some pattern in G_{gn}, and each G_g is similarly pruned using S_{sn}. Each set S_s therefore refers to the set $S_{s,gn}$ defined in section 4.1, and each G_g refers to $G_{sn,g}$. Pruning is done in this way in order to assure that the version space corresponding to every possible pairing of S_s and G_g is correctly represented.

5.4.2.2 An Example

The operation of the algorithm for updating boundary sets is illustrated here, *using the earlier feature interval learning example.* In this example, the sets S_0, S_1, G_0, and G_1 are to be determined. Figure 5.3 illustrates these four boundary sets derived from the same set of training instances used in figure 5.1. In order to make the drawing intelligible, only one pattern from each of these boundary sets is plotted in figure 5.3.

Notice the sets S and G in figure 5.1 are subsets of S_0 and G_0 in figure 5.3. The version space represented by S and G from figure 5.1 is the same as the one represented by S_0 and G_0 in figure 5.3. The reason G_0 contains concept descriptions in addition to those in G is that concept descriptions which do not match one positive instance are allowed in G_0, but not in G. These additional concept descriptions are included so that G_0 may be used in conjunction with S_1 to represent the version space $VS_{1,0}$. S and G may, of course be derived from S_0 and G_0 by removing from each set any concept descriptions not bounded by the opposite set in the partial ordering.



		G boundary set			S	S boundary set	
G0	g0:	3<x<11,	3<y<∞	S0	s0:	5≤x≤10,	5≤y≤11
		3<x<12,	3<y<13				
		3<x<14,	4<y<13				
		-∞<x<11,	6<y<∞				
		-∞<x<14,	6<y<13				
G1	g1:	-∞<x<12,	3<y<13	S1	s1:	5≤x≤9,	7≤y≤11
		-∞<x<11,	3<y<∞			7≤x≤10,	5≤y≤9
		-∞<x<14,	4<y<13				
		3<x<∞,	4<y<13				
		3<x<11,	-∞<y<∞				
		3<x<12,	-∞<y<13				
		3<x<12,	3<y<∞				
		3<x<14,	4<y<∞				
		3<x<14,	3<y<13				
		-∞<x<∞,	6<y<13				

Figure 5.3 Multiple boundary sets G0, G1, S0, and S1, for a consistent set of training instances.

Consider the performance of the algorithm described above if the negative instance (7, 10) is used to update the boundary sets illustrated in figure 5.3.

The boundary set G_1 is updated first. Initially, G_1 is the set of maximally general patterns consistent with any 4 of the 5 observed negative training instances. G_1 contains the following patterns:

```
 3<x<∞   4<y<13
-∞<x<∞   6<y<13
 3<x<11 -∞<y<∞
 3<x<12 -∞<y<13
-∞<x<11  3<y<∞
-∞<x<12  3<y<13
-∞<x<14  4<y<13
 3<x<12  3<y<∞
 3<x<14  4<y<∞
-∞<x<14  6<y<∞
 3<x<14  3<y<13
```

G_0, consistent with all 5 observed negative instances, is the set of patterns:

```
3<x<11  3<y<∞
3<x<12  3<y<13
3<x<14  4<y<13
-∞<x<11 6<y<∞
-∞<x<14 6<y<13
```

and S_1 (sn=1) is the set of patterns:

```
5≤x≤9   7≤y≤11
7≤x≤10  5≤y≤9
```

The set, U, of patterns minimally more specific than G_1 and consistent with the new negative instance is first determined.

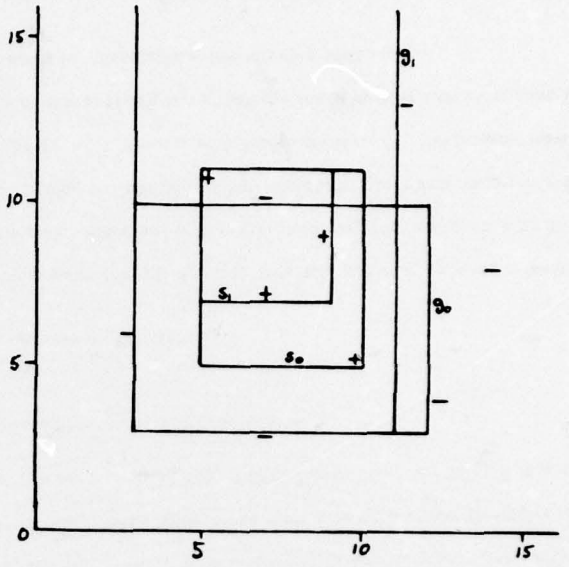
```
U = UPDATE-G(G1, S1, (7 10))

U:      3<x<∞   4<y<10
        3<x<12 -∞<y<10
       -∞<x<12  3<y<10
       -∞<x<14  4<y<10
        3<x<14  3<y<10   .
```

U contains the maximally general patterns consistent with any 4 of the original 5 negative instances, as well as the new instance. The boundary G_0 is consistent with all 5 of the original negative instances. The maximally general elements of the union of U and G_0 is therefore taken as the new boundary set G_1, shown below:

$3<x<\infty \quad 4<y<10$
$3<x<12 \quad -\infty<y<10$
$-\infty<x<12 \quad 3<y<10$
$-\infty<x<14 \quad 4<y<10$
$3<x<14 \quad 3<y<10$
$3<x<11 \quad 3<y<\infty$
$3<x<12 \quad 3<y<13$
$3<x<14 \quad 4<y<13$
$-\infty<x<11 \quad 6<y<\infty$
$-\infty<x<14 \quad 6<y<13$

The boundary set G_0 is updated next. Since the set G_{-1} is always empty, G_0 is updated as in the basic version space approach, without combining it with a second boundary set. After both G_1 and G_0 have been updated, the sets S_1 and S_0 are pruned by removing any pattern for which there is no more general pattern in G_1. In this case, no pruning takes place in either S_1 or S_0, and the resulting boundary sets are as shown in figure 5.4. Again, only a single pattern from each boundary set is plotted in this figure.



		G boundary set		S		S boundary set
G0	g0:	3<x<12,	3<y<10	S0	s0:	5≤x≤10, 5≤y≤11
		3<x<14,	4<y<10			
G1	g1:	3<x<11,	3<y<∞	S1	s1:	5≤x≤9, 7≤y≤11
		3<x<12,	3<y<13			7≤x≤10, 5≤y≤9
		3<x<14,	3<y<10			
		3<x<14,	4<y<13			
		3<x<12,	-∞<y<10			
		3<x<∞,	4<y<10			
		-∞<x<12,	3<y<10			
		-∞<x<14,	4<y<10			
		-∞<x<14,	6<y<13			
		-∞<x<11,	6<y<∞			

Figure 5.4 Multiple boundary sets from figure 5.3 updated in response to a new instance. An inconsistency arises with the introduction of this training instance.

5.4.3 An Optimal Solution to Learning with Limited Inconsistency

The extended version space approach described above finds all maximally consistent concept descriptions for concept learning problems involving a limited number of inconsistencies. If boundary sets S_0 through S_{sn} and G_0 through G_{gn} are initialized and modified as described above, then *all* maximally consistent concept descriptions will be found provided there is a consistent subset of at least t training instances, where t is the smaller of sn and gn. At least *some* maximally consistent concept descriptions will be found provided there is a consistent subset of at least p-sn positive instances and n-gn negative instances. These guarantees follow directly from the validity of representing $VS_{s,g}$ in terms of G_g and S_s, together with the correctness of the procedure for updating G_g and S_s with each training instance.

For the concept learning problem illustrated in figures 5.1, 5.3, and 5.4, only a single inconsistency occurs; that is, there is a consistent subset of all but one of the training instances. This inconsistency is detected when the negative training instance (7, 10) is presented. This instance forces the boundary G_0 to become more specific than S_0, indicating that the version space $VS_{0,0}$ is empty.

The task of the program then becomes to find all concept descriptions maximally consistent with the training instances - in this case, those consistent with any n+p-1 training instances.

In general, there may be many such consistent subsets of n+p-1 training instances. These will fall into two classes: (1) those consistent with p positive instances, and some n-1 negative instances, and (2) those consistent with some p-1 positive and n negative instances. The version spaces $VS_{0,1}$ and $VS_{1,0}$ respectively contain all concept descriptions consistent with these two classes of training sets. $VS_{0,1}$ contains concept descriptions consistent with the assumption that the inconsistency arose from a false negative instance, while $VS_{1,0}$ contains concept descriptions consistent with the assumption that the inconsistency was due to a false positive instance.

Notice that it is not possible to represent the union of $VS_{0,1}$ and $VS_{1,0}$ as a version space represented by a single pair of boundary sets. Therefore, unless it is known that the inconsistency arose from a false positive or from a false negative instance, the program must keep track of both $VS_{0,1}$ and $VS_{1,0}$ if it is to consider *all* maximally consistent concept descriptions[1].

5.4.3.1 A Note on Efficiency

Optimal solutions are feasible only for problems involving a limited number of inconsistencies. However, it is worth comparing this approach with the brute force method of finding the optimal solution in order to gain some insight into the efficiency of this approach to dealing with inconsistency. The discussion below concerning efficiency of representing and updating multiple version spaces applies as well to the heuristic approach discussed in the following section.

A brute force method for determining all maximally consistent concept

[1] In the Meta-DENDRAL program false positive instances are much more common than false negative instances. This is due to the way in which training instances are inferred from the available data, and because disjunctive concept descriptions must be learned.

descriptions is to determine the version space of each of the N subsets of N-1 training instances, then to take the union of these version spaces. Such an approach would have complexity N times the complexity of the basic version space approach. The extended version space approach, described above, obtains the same results as this brute force method, but requires only an additional comparison of two boundary sets per training instance beyond the computation required by the basic candidate elimination algorithm. This improvement in efficiency stems directly from generalizing the definition of version space to include concept descriptions consistent with alternate sets of training instances.

Some insight into the storage costs of representing $VS_{s,g}$ is provided by contrasting figure 5.1 with figure 5.3. The boundary sets shown in figure 5.3 are larger than those in figure 5.1, reflecting the increased cost of representing several version spaces consistent with a range of subsets of the training instances. One might expect that to represent $VS_{s,g}$ should require the storage needed to represent a version space of any particular subset of training instances, multiplied by the number of possible subsets of p-s positive instances and n-g negative instances. In the current example, there are 20 possible subsets of training instances corresponding to $VS_{1,1}$. Thus, since the version space corresponding to the single set of all training instances in figure 5.1 is represented by 4 patterns, one might expect approximately 80 patterns would be required to represent $VS_{1,1}$ in figure 5.3. However, only 13 patterns are required to represent $VS_{1,1}$.

Two main factors contribute to the smaller than predicted sizes of the boundary sets which represent $VS_{s,g}$. First, only a small proportion of the possible subsets of

training instances corresponding to $VS_{s,g}$ are consistent with respect to the concept description language. In the current example, only 2 of the 20 subsets were consistent. Thus, many possible subsets contribute no concept descriptions to $VS_{s,g}$. Second, of those subsets of training instances which are consistent, the version spaces will in general overlap. Thus, storage requirements do not increase linearly even with the number of consistent subsets of training instances corresponding to $VS_{s,g}$.

It is also of interest to examine the number of patterns contained in the boundary sets over the course of processing training instances. This data is summarized in table 5.1 below[1]. Notice that the size of each boundary set tends to increase up to a certain limit as training instances are processed. The sizes of these sets then tend to remain at this limit, although the sizes vary considerably from step to step. This trend is typical of the set sizes for both the basic version space approach and the extended approach. The same general behavior has been recorded for multiple boundary sets in the Meta-DENDRAL domain, in which concepts are described in a structural pattern language.

[1] Note that for the feature interval learning problem, the S_0 boundary set must always contain a single pattern (the intervals delimited by the maximum and minimum observed values of each feature).

Training Instance Sequence		Sizes of Version Space Boundary Sets			
		S_1	S_0	G_0	G_1
(7, 7)	+		1	1	
(5, 11)	+	2	1	1	
(14, 8)	-	2	1	3	1
(7, 3)	-	2	1	4	3
(3, 6)	-	2	1	6	7
(11, 13)	-	2	1	8	12
(10, 5)	+	2	1	5	10
(12, 4)	-	2	1	6	13
(9, 9)	+	2	1	5	11
(7, 10)	-	2	1	2	10

Table 5.1. Sizes of version space boundary sets at each step in the training sequence. Data is taken from the example shown in figure 5.3. positive instances are indicated by a "+", negative instances by a "-".

5.5 A Heuristic Approach to Learning with Multiple Inconsistencies

The approach assured to find an optimal solution is feasible only for problems involving a limited number of inconsistencies. If even a moderate number of training instances must be rejected to arrive at a consistent subset of instances, the approach becomes inefficient. However, the procedure for obtaining the optimal solution leads directly to a heuristic approach for dealing with learning problems involving many inconsistencies.

The heuristic approach to dealing with inconsistency requires a criterion for selecting the "best" version space within the program's consideration at each step, and two parameters, sn and gn, which limit the number of S_s and G_g boundary sets respectively which the program is to consider. After each training instance, the best version space, $VS_{s,g}$, is chosen from those which the program has represented. The boundary sets S_s and G_g which represent this version space are retained, as are S_{s+1} through S_{s+sn}, and G_{g+1} through G_{g+gn}. Any other boundary sets are dropped from consideration. If any of the boundary sets to be retained are not currently stored by the program, they are approximated using a procedure described below.

Since the version space $VS_{s,g}$ contains all concept descriptions consistent with any subset of p-s positive and n-g negative instances, the criterion for ranking version spaces corresponds to a criterion for selecting among alternate classes of consistent subsets of training instances.

Reasonable criteria for selecting the central version space are to choose $VS_{s,g}$ such that s is minimized, g is minimized, or s+g is minimized. Minimizing s corresponds to selecting the version space consistent with the greatest number of positive instances. Minimizing g corresponds to selecting the version space consistent with the greatest number of negative instances. Minimizing s+g corresponds to selecting the version space consistent with the largest possible number of training instances without distinguishing between positive and negative instances.

5.5.1 Creating New Boundary Sets

When the heuristic procedure selects a current best version space and the associated range of boundary sets S_s through S_{s+sn} and G_g through G_{g+gn}, not all of the required boundary sets may be currently stored by the program. Any such boundary sets are approximated by the program as described in this section.

Suppose that a sequence of boundary sets up through S_s is currently kept by the program, and that the additional boundary set S_{s+1} is desired. The set S_s is the set of maximally specific concept descriptions consistent with some p-s of the p observed positive training instances. When the next positive instance is encountered, S_{s+1} will be defined as the current set S_s, since at that point the set will be consistent with p-s = (p+1)-(s+1) of the (p+1) observed positive instances.

The boundary set G_{g+1} is created from the set G_g in an analogous manner, when the next negative training instance is observed. In general, the sets S_{s+i} and G_{g+i} are approximated by the current sets S_s and G_g once i additional positive or negative training instances have been encountered.

The above procedure gives only an approximation of the true set S_{s+i}. By definition, S_{s+i} is the set of maximally specific concept descriptions consistent with *any* p-(s+i) of the p observed positive instances. Although the above procedure correctly determines concept descriptions consistent with this number of positive instances, it does not consider *all* possible subsets of p-(s+i) positive instances. In particular, since S_{s+1} is set to the value of S_s when a new positive instance is encountered, any subset of positive instances including the new positive instance is

not considered at this point. Depending upon whether these ignored subsets would or would not have been consistent with more specific concept descriptions than those already in S_s, the approximation of S_{s+1} will not or will be accurate.

5.5.2 Limitations

Although the heuristic approach described here attempts to find concept descriptions maximally consistent with the training data, it does not always succeed. The approach may be characterized as one which considers a range of version spaces corresponding to a range of consistent subsets of training instances at each step. Since the program can consider only some of the possible consistent subsets of instances, it chooses a class of the largest such subsets (corresponding to the selected version space), and several classes of progressively smaller subsets (corresponding to the version spaces represented by the additional boundary sets). The program can find maximally consistent concept descriptions within the scope of the consistent subsets of training instances considered at each step. The performance of the program therefore is determined by the scope of consistent subsets of instances which the program chooses to consider.

This heuristic method differs from an exact method in that it considers only some of the possible consistent subsets of training instances. This incompleteness is due to two factors: (1) at any step in the process there may be several classes of largest consistent subsets of training instances (i.e., $VS_{2,0}$, $VS_{1,1}$, $VS_{0,2}$), but the program chooses only one of these; and (2) creating new boundary sets is an approximate procedure.

The first of these two limitations on completeness in the heuristic approach may be overcome in exchange for a different shortcoming. Rather than select among alternate plausible version spaces, the program might instead choose the smallest version space which contains all these version spaces. For example, rather than choose between $VS_{2,0}$, $VS_{1,1}$, and $VS_{0,2}$, select the version space $VS_{2,2}$ which contains each of these possibilities. This strategy allows the program to retain all maximally consistent concept descriptions, but results in additional concept descriptions, consistent with fewer training instances, being considered as well ($VS_{2,2}$ contains more concept descriptions than the union of $VS_{2,0}$, $VS_{1,1}$, and $VS_{0,2}$). If the cost of obtaining training instances is low, this may be a good strategy, since additional *training instances* may be relied upon to eliminate the extra concept descriptions from $VS_{2,2}$.

The heuristic approach described here is applied in the following section to a concept learning problem in which inconsistency arises from the fact that a disjunction of available concept descriptions is needed to cover the entire set of training instances.

5.6 Learning Disjunctive Concepts

One cause of inconsistency discussed in section 5.2.3 is that a disjunctive set of concept descriptions may be required in order to cover all observed training instances. Here the use of the extended version space approach for learning

disjunctive concept descriptions is illustrated. Of course, the version space approach may be applied to concept learning problems in which the concept description language itself allows disjunctions (resulting in larger boundary sets than if the language does not allow disjunctions).

In this section, the extended version space approach is used as a building block in a simple iterative approach to learning disjunctive concepts. The training instances are first processed to search for the version space corresponding to the largest consistent subsets of the training instances. A concept description from this version space is then used to filter out positive training instances which it matches, and the procedure iterates on the remaining unexplained training instances. The process continues until all training instances are explained by some concept description.

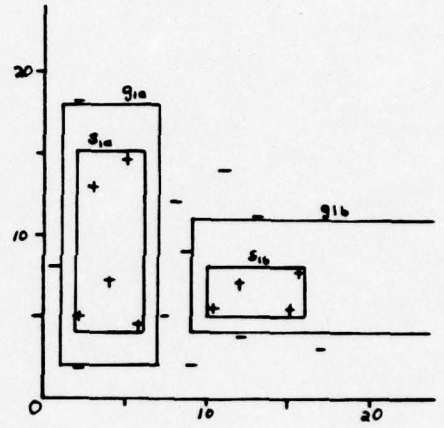
Notice that although this iterative approach will not in general find the smallest set of disjuncts required to explain the data, it tends to find the disjuncts in order of the number of training instances which they explain. Thus, if it is desirable to set a threshold on the minimum number of training instances which a disjunct must uniquely cover in the training data, this process may be terminated when a disjunct is generated which does not meet the threshold.

The heuristic approach described in the previous section is used here. In this example, the strategy for selecting the current best $VS_{s,g}$ is as follows:

> *Version Space Selection Strategy for Learning Disjunctive Concepts:* At each step select the version space consistent with all negative instances and the largest possible number of positive instances. In other words, select $VS_{s,g}$ for which g=0 and s is minimized.

During each iteration through the data, the program learns a single disjunct in the eventual disjunctive concept description. It is expected, therefore, that during each iteration except the last, a set of positive instances which cannot be completely covered by a single disjunct will be presented. For this reason the above selection strategy selects the $VS_{s,0}$ which is consistent with *every* negative training instance, but which may be inconsistent with some positive instances. In this example, the parameters controlling the number of additional boundary sets kept by the program are set to sn=2 and gn=0.

Figure 5.5 illustrates a set of training instances, and the version spaces of the two disjuncts which were found by the program to cover these instances.



		G boundary set	S boundary set
GA	g1a:	1<x<7, 2<y<18	SA s1a: 2≤x≤6, 4≤y≤15
GB	g1b:	9<x<∞, 4<y<11 7<x<∞, 4<y<9	SB s1b: 10≤x≤16, 5≤y≤8

Figure 5.6 Learning a disjunctive concept.

During the first iteration through the training instances, the program found the version space containing the concept description (1<x<7, 2<y<18). The five positive instances which this concept description covers were then removed from the training data, and the procedure iterated on the reduced set of training instances. During the second iteration, the program found the version space containing the pattern (9<x<∞, 4<y<11), which covers the remaining three training instances. Thus, in two passes through the training data, the program discovered the disjunctive concept description (1<x<7, 2<y<18) OR (9<x<∞, 4<y<11), consistent with all training instances.

5.7 Summary

Inconsistency can arise in concept learning problems from several causes. The extended version space approach of representing and updating multiple version spaces, each consistent with a class of consistent subsets of the training instances, is presented in this chapter. This extended approach is capable of determining all concept descriptions maximally consistent with the training data in learning problems involving a limited number of inconsistencies. A suboptimal (but less costly) heuristic approach is presented for concept learning problems involving a larger number of inconsistencies.

Chapter 6

Version Spaces and Meta-DENDRAL

6.1 Introduction

One of the best measures of the utility, capabilities, and limitations of version spaces is provided by their application to a concept learning problem in science. The *version space approach* to concept learning has been implemented as one part of the Meta-DENDRAL program [Buchanan, 1978] which learns production rules in the domain of chemical spectroscopy. These rules predict major peaks in the mass spectra and ^{13}C NMR spectra of classes of organic molecules.

In this chapter, the problem of inferring rules of mass spectroscopy and ^{13}C NMR spectroscopy is briefly described. This rule inference problem is broken down into two major tasks: (1) *determining a set of training instances* for this rule learning task from the available training *data*, and (2) learning rules from these training instances. The Meta-DENDRAL learning problem is complicated by noisy, ambiguous data, the need to learn a disjunctive set of rules, and the known insufficiency of the concept *description language*.

The use of version spaces in Meta-DENDRAL is described. The concept learning problem is outlined in the terms used in earlier chapters, and the algorithm for updating the version space boundaries is described in detail. Particular aspects of the problem domain which influence the use, performance, and efficiency of the version space

approach are discussed, as well as intuitions concerning the general viability of the *version space approach*, gained from experimenting with this implementation.

6.2 The Chemistry Problem

The Meta-DENDRAL problem is to discover general rules within a prescribed language for predicting peaks in the mass spectra and ^{13}C NMR spectra of classes of molecules. These rules are inferred from training data consisting of pairs of known molecules and their associated spectra. The rules generated by Meta-DENDRAL are useful both to chemists for the insight which they provide, and as part of the knowledge base of a set of computer programs called DENDRAL [Feigenbaum, 1971]. The DENDRAL programs apply knowledge of several areas of chemistry to the problem of elucidation of molecular structures from chemical data.

The *rules which Meta-DENDRAL learns are* of the form (pattern → action). These are interpreted as follows: if the pattern matches a molecule, then the specified action is taken. In Meta-DENDRAL the pattern characterizes a substructure within a molecule, and the action specifies the appearance of spectral peaks in some region of the spectrum[1]. When the pattern fits within a molecule, the *corresponding spectral* peaks are predicted for that molecule.

The Meta-DENDRAL program is described in [Buchanan, 1978]. In the following subsections, the two areas of spectroscopy are briefly described, and their common concept learning problem described in the terminology of earlier chapters. Some

[1] *In the mass spectrometry* domain the rules specify processes which produce spectral peaks rather than the peaks themselves.

simplifications of the chemical problem have been made in the following discussion in order to allow presenting the main ideas compactly.

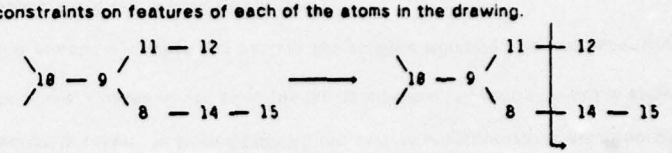
6.2.1 Mass Spectrometry

A mass spectrometer is an instrument for chemical analysis which produces a characteristic spectrum (set of peaks) from a small sample of any given compound. This spectrum is useful for structure elucidation, since it acts as a "fingerprint" to determine the identity of the chemical sample used to generate the spectrum. Chemists are interested in being able to predict the mass spectrum of a given molecule, and, more importantly, in being able to reconstruct the molecular structure of an unknown compound from its mass spectrum.

The mass spectrometer bombards a chemical sample with a beam of electrons which causes molecules in the sample to break apart. The mass of resulting molecular fragments is then detected by the machine. A spectrum is produced in which the mass of each detected fragment is indicated by a spectral peak. The intensity of each mass peak is determined by the relative number of fragments of that mass which were detected by the machine.

Meta-DENDRAL forms rules that predict which bonds in a given molecule can be expected to break, and therefore yield peaks in the mass spectrum of the molecule. A typical rule of mass spectroscopy generated by Meta-DENDRAL is shown in figure 6.1. This rule indicates that if the substructure described on the left appears within a molecule, then the bonds between atoms 8 and 14, and between atoms 11 and 12 will

break in a mass spectrometer. The bar through these bonds indicates that they will break, and that a mass spectral peak will be observed at the mass of the right hand fragment[1] resulting from the break. The table below the drawing describes constraints on features of each of the atoms in the drawing.



NODE	NODE CONSTRAINTS			
	Atom Type	Non-Hydrogen Neighbors	Hydrogen Neighbors	Unsaturated Electrons
8	any	≥2	any	any
9	any	3	any	any
10	any	≥3	any	any
11	any	≥2	any	any
12	any	≥1	any	any
14	any	≥2	any	any
15	any	≥1	any	0

Figure 6.1 – A MetaDENDRAL mass spectroscopy rule

In order to obtain training instances from which to determine such rules, Meta-DENDRAL must first determine which bonds in each training molecule have broken to yield the fragment masses represented in the mass spectrum. Since there are usually many ways to break up a molecule to obtain fragments of a given mass, the mapping of spectral peaks to the molecular fragmentations which cause them is ambiguous. General knowledge of mass spectroscopy is therefore brought to bear during this step

[1] This convention reflects the fact that only positively charged fragments are detected. The arrow in the figure indicates which fragment is charged.

as described in [Buchanan, 1978]. The result of this procedure, implemented in the INTSUM (INTerpret and SUMmarize data) program, is a set of partly reliable training instances corresponding to sets of bonds in the training molecules which are expected to have broken (positive instances) and not to have broken (negative instances) in the mass spectrometer. Each set of bonds capable of cutting a molecule is therefore a positive or negative instance for the rule whose action is "break these bonds".

6.2.2 C13 NMR Spectrometry

^{13}C nuclear magnetic resonance spectroscopy (^{13}C NMR) is a second spectroscopic technique for which Meta-DENDRAL forms rules [Mitchell, 1978]. As with mass spectrometry, the ^{13}C NMR spectrum provides information about the molecular structure of an unknown compound. ^{13}C NMR spectroscopy takes advantage of the fact that the nucleus of ^{13}C, a naturally occurring isotope of carbon, resonates when placed in a strong magnetic field, at a frequency which is influenced by the local environment of the ^{13}C atom. Therefore, in a given molecule, each ^{13}C atom will have a resonance frequency depending upon its environment within that molecule. The ^{13}C NMR spectrometer detects the resonance frequencies of these nuclei by bombarding the sample with electromagnetic radiation, and measuring frequencies at which this radiation is absorbed. The ^{13}C spectrum is a list of observed frequencies at which radiation is absorbed, indicating the resonance frequencies of the various ^{13}C atoms in the sample.

The rules formed by Meta-DENDRAL for predicting ^{13}C NMR spectra are similar

to those formed in the mass spectrometry domain. They are also of the form (pattern → action), in which the pattern again characterizes some molecular substructure, and the action is the prediction of a spectral peak in a given region of the ^{13}C NMR spectrum. Figure 6.2 shows a typical ^{13}C NMR rule generated by Meta-DENDRAL. This rule predicts that the nucleus of atom 1 is associated with a peak in the indicated region of the spectrum (measured in parts per million from a standard reference peak).

1 — 2 — 3 — 4 ⟶ 14.8 ppm. ≤ peak(1) ≤ 14.7 ppm.

NODE CONSTRAINTS

NODE	Atom Type	Non-Hydrogen Neighbors	Hydrogen Neighbors	Unsaturated Electrons
1	^{13}C	1	any	any
2	C	2	any	any
3	C	2	any	any
4	C	≥1	2	any

Figure 6.2 – A MetaDENDRAL 13C NMR rule

There is an important difference in the training information required by Meta-DENDRAL for the ^{13}C NMR problem and the mass spectroscopy problem. As mentioned earlier, the training data for mass spectroscopy are pairs of known molecules and observed spectra, from which training instances are determined by the INTSUM program. There is as yet no automated analog to INTSUM for the ^{13}C NMR domain, so the chemist must indicate which ^{13}C atoms in the training molecule give rise to each observed spectral peak.

6.3 The Concept Learning Problem

For both of the above domains, a major part of the rule learning problem is to find generalizations from the training set of molecular structures. The generalization problem is one of finding a chemical substructure which characterizes those training instances for which a given set of spectral peaks appears. For the mass spectroscopy problem, chemical substructures must be found to characterize the important local chemical environment surrounding bonds that break in a mass spectrometer. For the ^{13}C NMR problem, substructures must be found to characterize the local environment surrounding ^{13}C atoms whose nuclei resonate at or near a given frequency.

Finding generalizations of the molecular sites described by training instances is one part of rule learning which is a concept learning problem in itself, and is the part of the rule learning process for which version spaces are useful.

Various uses of version spaces within Meta-DENDRAL have been considered, all of which involve the use of version spaces in the context of the following concept learning problem:

MetaDENDRAL Concept Learning Problem:

Given: 1. *Concept description language:* All possible networks with nodes representing non-hydrogen atoms in a molecule, and links between nodes representing chemical bonds between the corresponding atoms. Each node (atom) has the following properties which may be constrained as shown:

Property	Allowed Constraints
Atom-Type	C, N, O, S, ..., any
Non-Hydrogen-Neighbors	0, ≥0, 1, ≥1, 2, ...
Hydrogen-Neighbors	0, 1, 2, ..., any
Unsaturated-Electrons	0, 1, 2, ..., any

Each link in the network represents a chemical bond between the corresponding non-hydrogen atoms. An absent link indicates there cannot be a chemical bond between the corresponding atoms. Each pattern has an associated *site* which is a particular atom (in the NMR domain) or set of bonds (in the mass spectroscopy domain) with which the spectral peak is associated. A pattern may contain only nodes that are directly or indirectly connected to the site of the pattern.

2. *Pattern Matcher:* A pattern matches a training instance if the pattern constraints are satisfied by the instance; that is, if there is a mapping of pattern nodes into instance nodes such that (1) the node or links in the pattern site map into the node or links in the instance site, (2) the pattern constraints for each node property are satisfied, (3) nodes connected by a link in the pattern are mapped to nodes connected by a link in the instance, and (4) nodes not connected by a link in the pattern are mapped to nodes not connected by a link in the instance.

3. *Training instances:* Network descriptions of molecules with associated sites, labeled as either positive or negative instances of the target concept. Each feature of each node in the molecule must have a specific value (i.e., Atom-Type may be "C", but may not be "any").

Determine: All concept descriptions within the given language which are consistent with the training instances.

The pattern language, pattern matcher, training instances, and general-to-specific partial ordering are discussed further in the following sections.

6.3.1 The Pattern Language

Each pattern in the Meta-DENDRAL program describes a chemical substructure which may occur within a molecule, and locates a particular site (atom or set of bonds) within that chemical substructure. The site corresponds in the ^{13}C NMR and mass spectroscopy domains, respectively, to the atom or set of bonds within the molecule which yield the predicted spectral peak. Allowed patterns are restricted to those which contain only atoms connected to the pattern site by some chain of chemical bonds.

The pattern language described above is a structural, or network language which states constraints on several node properties, and which specifies chemical bonds as links between the nodes. Hydrogen atoms are the only atoms not explicitly represented by nodes. Because they are so common, they are represented instead by the Hydrogen-Neighbors feature of each node in the pattern. This language is similar to the structural language described in chapter 2 for the arch learning problem. Figures 6.1 and 6.2 illustrate legal patterns in this language.

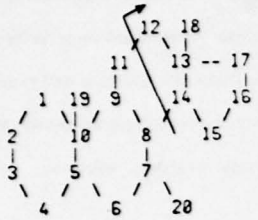
The node features Atom-Type, Non-Hydrogen-Neighbors, Hydrogen-Neighbors, and Unsaturated-Electrons are interdependent; that is, given specific values for Atom-Type and any two other features, the fourth feature is uniquely determined. As a result, it is possible to construct syntactically distinct, but semantically equivalent patterns which match exactly the same instances in the space of possible instances. Node constraints implied by other node constraints are therefore made explicit in each pattern in the general boundary of the version space, in order to keep this

redundancy in the pattern language from adversely affecting the size of the boundary sets.

It should be noted that although chemically meaningless patterns may be constructed in the language of chemical substructures (i.e., a substructure containing an atom with 20 neighbors), the program will never construct version spaces containing such patterns. Only patterns which match the observed positive training instances are allowed members of the version space. Thus, only patterns which are substructures of observed (and hence realizable) structures are considered.

6.3.2 The Instances

The language of training instances is a subset of the pattern language. Each training instance is a complete molecule with an associated site. A training instance for the mass spectroscopy problem, consisting of a complete molecule and the indicated broken bonds (the site), is shown in figure 6.3.



break bonds between atoms 11 and 12, and 8 and 14

mass peak observed at mass corresponding to fragment containing atom 12

NODE	NODE CONSTRAINTS			
	Atom Type	Non-Hydrogen Neighbors	Hydrogen Neighbors	Unsaturated Electrons
1	C	2	2	0
2	C	2	2	0
3	C	2	2	0
4	C	2	2	0
5	C	3	1	0
6	C	2	2	0
7	C	3	1	1
8	C	3	1	0
9	C	3	1	0
10	C	4	0	0
11	C	2	2	0
12	C	2	2	0
13	C	4	0	0
14	C	3	1	0
15	C	2	2	0
16	C	2	2	0
17	C	2	2	0
18	C	1	3	0
19	C	1	3	0
20	O	1	0	1

Figure 6.3 - A Training Instance for Mass Spectrometry Rules

6.3.3 The Pattern Matcher

It is simplest to define the pattern matching predicate by first defining the concept of a connected mapping of pattern nodes to instance nodes. Given a mapping, X, of pattern nodes to instance nodes, we use the notation X(p) to refer to the instance node to which the pattern node p is mapped.

Connected Mapping: A mapping, X, from pattern nodes to instance nodes is a connected mapping if (1) the mapping is 1-to-1 and *into*, and (2) every pair of instance nodes $X(p_1)$ and $X(p_2)$ share a common link if and only if the pattern nodes p_1 and p_2 also share a common link.

A pattern matches an instance if there is a connected mapping from pattern nodes to instance nodes for which all the node constraints in the pattern are satisfied.

Pattern Matcher: A pattern P matches an instance I if there is a connected mapping, X, of nodes from P into nodes from I such that for each pattern node, p, each feature *constraint* stated in p is satisfied by the corresponding feature *value* in the instance node X(p).

6.3.4 The Partial Ordering

The general definition of the general to specific ordering, first stated in chapter 2, is repeated here.

Pattern P1 is ***more specific than or equal to*** pattern P2 (P1 ≥ P2) if and only if P1 matches a subset of all the instances which P2 matches.

This definition, together with the above definition of the pattern matcher yields the following equivalent, but domain specific, definition of the partial ordering for the current pattern language.

Domain Specific Definition of Partial Ordering: Pattern P1 is ***more specific than, or equal to*** pattern P2 if and only if there is a connected mapping, X, of nodes in P2 into nodes in P1 such that for each pair of nodes p_2, $X(p_2)$, the feature constraints associated with $X(p_2)$ are more specific than or equal to the feature constraints associated with p_2.

The general-to-specific ordering on patterns may therefore be expressed in terms of a general-to-specific ordering on node feature constraints. The ordering on the node constraints is simple (e.g., the constraint "C" for the feature Atom-Type is more specific than the constraint "any"). Figure 6.4 illustrates the partial ordering for constraints on the node feature Non-Hydrogen-Neighbors.

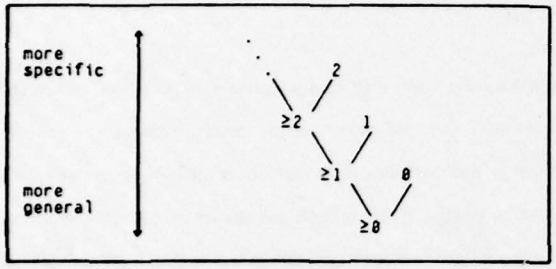


Figure 6.4: Partial ordering for constraints on the node feature Non-Hydrogen-Neighbors

6.3.5 Inconsistencies

Meta-DENDRAL faces several sources of inconsistency in the set of training instances associated with the above concept learning problem. In the mass spectroscopy domain, training instances for the above concept learning problem are not directly available but must be determined from the training data as noted earlier.

But the data from which training instances are determined are not totally reliable, due to noise peaks or missing peaks in the spectrum. In addition, the language of chemical substructures is known to be insufficient to capture all possibly relevant features of chemical substructures. Chemists themselves do not know all the correct features. Finally, this is a disjunctive concept learning problem. There is no single characteristic substructure which covers all instances in which molecules fragment, or all instances in which a ^{13}C NMR resonance peak occurs in a given region of the spectrum. Instead, many rules are required to explain a substantial portion of the training data.

Because of these inconsistencies a single rule of mass spectroscopy typically covers at most 5% of the positive training instances proposed by INTSUM. Although the approach described in chapter 5 for dealing with inconsistent learning problems is implemented in Meta-DENDRAL, this approach cannot efficiently deal with such widespread inconsistency. As a result, the inconsistencies in the INTSUM proposed training instances must be reduced in some way before the version space approach may be practically applied to the problem. Methods for reducing this inconsistency are discussed in a later section of this chapter. Before considering this problem further, we describe the algorithm for concept learning using version spaces in Meta-DENDRAL.

6.4 The Learning Algorithm

The candidate elimination algorithm which uses version spaces for concept learning is described in domain independent terms in chapter 2. Chapter 6 describes

an extended algorithm for dealing with inconsistent learning problems. The extended algorithm of chapter 5 is implemented in **Meta-DENDRAL**, where it is used to learn rules from uncertain training instances. The number of boundary sets used by the extended algorithm is under user control, as are optional heuristics for pruning these boundary sets if they surpass a user set threshold.

This section describes the implementation of the functions UPDATE-S and UPDATE-G for the **Meta-DENDRAL** pattern language, as well as the use of optional heuristics and domain knowledge to prune the version space boundary sets. UPDATE-S(S, G, i) is the routine which determines the updated S boundary given a new positive instance, while UPDATE-G(G, S, i) determines the updated G boundary set given a new negative instance. Those readers who are not interested in the implementation of these two functions may want to skip over the two subsections 6.4.1 and 6.4.2.

The definitions of UPDATE-S and UPDATE-G are repeated below from chapter 2. P refers to the set of all possible patterns, and VS refers to the current (not yet updated) version space.

$$\text{UPDATE-G}(G, S, i) = \text{MIN}(\{p \in VS \mid \neg M(p,i)\})$$
$$= \text{MIN}(\{p \in P \mid (\exists s \in S)(\exists g \in G)((s \geq p \geq g) \wedge \neg M(p,i))\})$$

$$\text{UPDATE-S}(S, G, i) = \text{MAX}(\{p \in VS \mid M(p,i)\})$$
$$= \text{MAX}(\{p \in P \mid (\exists s \in S)(\exists g \in G)((s \geq p \geq g) \wedge M(p,i))\}).$$

As discussed in chapter 2, the boundary sets S and G trace out complementary

breadth-first searches through the space of allowed concept descriptions. It is possible to implement UPDATE-S and UPDATE-G as simple generate and test procedures which consider progressively less specific (less general) patterns along each branch of the partial ordering until a pattern is found which is consistent with the new instance. The new boundary set could then be found by taking the maximally specific (maximally general) patterns found during this search, and removing those patterns which are not bounded in the partial ordering by some pattern in the opposing boundary set. Such procedures would, however, be inefficient because they require explicitly examining every pattern between the old and new boundary sets.

It is, fortunately, possible to improve substantially on the above brute force methods for implementing the functions UPDATE-G and UPDATE-S. By considering prospectively (1) which patterns in S and G lie along the same branch in the partial ordering, and (2) possible mappings of patterns in S to the new instance, it is possible to move directly to the the correct pattern along each branch of the partial ordering, without explicitly considering intermediate patterns.

6.4.1 Updating the Specific Version Space Boundary Set

Functions similar to UPDATE-S have been implemented by several researchers for pattern languages in several domains. Plotkin's least generalization algorithm [Plotkin, 1971], Hayes-Roth's interference matching algorithm [Hayes-Roth, 1975], and Vere's maximal common generalizations algorithm [Vere, 1975] are all similar in intent to UPDATE-S. All of these approaches involve some method for considering possible correspondences or mappings from the pattern to the instance, then

determining generalizations of the pattern which allow it to match the instance according to one of these mappings. Several aspects of this "partial matching" problem are described by Hayes-Roth in [Hayes-Roth, 1978].

The implementation of UPDATE-S in Meta-DENDRAL is similar to the algorithms mentioned above. The program considers patterns in the set S one at a time, as described below, to find minimally more general patterns consistent with the positive instance. The variable NEW-S is used to store the new boundary set S as it is being constructed.

```
UPDATE-S(S, G, i)

FOR EACH pattern, p, in S,
    IF p matches i, THEN add p to NEW-S and delete from NEW-S
                         any pattern more general than p.
    ELSE BEGIN
        Consider all maximal connected mappings of subpatterns of p to i.
        For each such mapping consider all least generalizations of nodes in
              p, including the removal of nodes from p, which are required
              to allow p to match i according to this mapping.
        For each resulting pattern, if the pattern is less specific than
              some pattern in NEW-S, or if there is no pattern in G
              more general than this pattern, then ignore the pattern,
              else add the pattern to NEW-S, and delete from NEW-S any
              pattern which is more general than this pattern.
    END;
```

The term *maximal connected mappings of subpatterns of p to i* refers to connected mappings of subsets of connected nodes of p into I which are not contained in any other mapping. These mappings are determined by initially mapping the site of the

pattern to the site of the instance, then repeatedly mapping neighbors of mapped nodes to each other.

6.4.2 Updating the General Version Space Boundary Set

The procedure for updating the general version space boundary takes advantage of the fact that the patterns in the S boundary provide a complete "menu" of constraints which may be added to patterns in G. In other words, *only* by adding constraints from some pattern in S can the resulting pattern be a member of the current version space, and therefore be a valid member of the G boundary set.

The implementation in Meta-DENDRAL of the function UPDATE-G is described below. By determining all ways (alternate mappings) in which a general pattern in G is more general than a specific pattern in S, the program determines all ways in which nodes in the general pattern may be made more specific. The program then finds all possible mappings of the general pattern to the negative instance, and adds to the pattern the minimal set of constraints needed to assure that each mapping is not a valid match. Possible mappings of the general pattern to the instance are determined by finding maximal connected mappings of subpatterns of the specific pattern to the instance. These mappings include all connected mappings of the general pattern to the instance, as well as mappings of nodes which might be added to the general pattern.

```
UPDATE-G(G, S, i)

FOR EACH pattern, p, in G, determine all mappings of p to patterns in S.
FOR EACH maximal connected mapping, m, of patterns, s, in S to i,
    BEGIN
        FOR EACH pattern, p, in G,
            IF p matches i by the composition of some mapping of p to s, and
                some mapping of s to i,
            THEN replace p in G by the set of minimal patterns derived
                by adding constraints from s to p so that p no longer
                matches i under this mapping.
            Remove all patterns from G which are not maximally general.
    END.
```

6.4.3 Use of Domain Knowledge and Heuristics

Knowledge of chemistry may be employed by the learning algorithm in two ways. First, because the language of chemical substructures allows syntactically distinct, but semantically equivalent patterns, knowledge of the meaning of these patterns is necessary for deleting redundant patterns from the version space boundaries. This use of domain knowledge has no effect on the completeness of the version space approach. Second, because the version space boundaries become quite large for some problems, it is sometimes useful to apply heuristics based on outside knowledge to prune the boundaries. If heuristic pruning is used, of course, the program can no longer be assured to determine *all* concept descriptions consistent with the training

instances. The remainder of this section describes domain knowledge and heuristics which the user may apply to the candidate elimination algorithm.

Knowledge of the interdependency of the node features is always used by the program. As described in section 6.3.1, values for some subsets of node features completely determine the value for the remaining feature. For example, if a given node in some pattern has the following node constraints,

```
Atom-Type = C
Non-Hydrogen-Neighbors = 3
Hydrogen-Neighbors = any
Unsaturated-Electrons = 0
```

then any atom which matches this pattern node must have 1 hydrogen neighbor (since carbon has a valance of 4). This node will match exactly the same instance nodes as the following node.

```
Atom-Type = C
Non-Hydrogen-Neighbors = any
Hydrogen-Neighbors = 1
Unsaturated-Electrons = 0
```

Because these two sets of node constraints have the same chemical meaning, and because neither is more general than the other, if there is a pattern in the boundary set G which contains one of these nodes, there will be a second pattern identical to the first, but containing the other node. This leads to a multiplication of the size of the general boundery set. In order to prevent this occurrence, when a new pattern is generated in the general boundary, any node constraints which are implied by stated node constraints are made explicit in the node. Thus, if either of the above nodes appeared in a pattern in the general boundary, that node would be replaced by the following node.

```
Atom-Type = C
Non-Hydrogen-Neighbors = 3
Hydrogen-Neighbors = 1
Unsaturated-Electrons = 0
```

The procedure which makes these implicit node constraints explicit uses knowledge of the valances of atoms, and of the interdependence of the node features. This use of domain knowledge eliminates redundant patterns from the boundary set, thereby improving efficiency without affecting the completeness of the learning algorithm.

In contrast to the above use of domain knowledge, there are several methods available for pruning the boundary sets which may affect the completeness of the algorithm. These options are listed and explained below.

Imposing assumptions concerning the class of molecules considered: Meta-DENDRAL forms rules valid for *classes* of compounds. Simple assumptions concerning the class of structures under study may sometimes lead to effective pruning of the version space boundaries. The user has the option of adding two additional assumptions to the routine described above for adding implicit node constraints. (1) By assuming a maximum valance for atoms considered within the class of compounds, additional node constraints will be filled in by the above procedure. (2) The user may also specify that all atoms with a valance of four are carbon atoms. This is often the case in organic molecules, and allows the program to fill in additional node constraints (Atom-Type) for certain nodes. Both of these assumptions, *when correct*, allow pruning the version space without endangering the completeness of the candidate elimination algorithm.

Weakening the pattern language: The user may elect to consider only a subset of the node features in the pattern language when updating the general boundary set. This option allows choosing a more limited pattern language, and the operation of the program may then be characterized as a complete examination of the reduced language of rules. When operating with a reduced pattern language, the user may instruct the program to begin considering the entire range of node properties when the selected subset is not sufficient to find patterns consistent with all instances.

Limiting branching of the boundary sets: There are several ways in which the branching of the specific and general boundary sets may be reduced heuristically. The user may set a parameter which limits the number of alternate mappings of patterns to instances considered in several portions of the procedures UPDATE-S and UPDATE-G. Limiting the number of mappings considered may or may not reduce the branching of the boundary sets, and usually reduces the time required by UPDATE-G and UPDATE-S.

Setting a ceiling on sizes of boundary sets: If, after other pruning of the boundary sets, the size of the set still exceeds some user defined threshold, the sets may simply be truncated. The specific boundary set is truncated to the specified size by deleting the patterns which contain the fewest nodes, while the general boundary is truncated by removing the patterns which contain the largest number of nodes.

6.5 Using Version Spaces in Meta-DENDRAL

How is the version space approach used in conjunction with other parts of the

Meta-DENDRAL program? What new capabilities derived for Meta-DENDRAL by the use of version spaces?

The version space approach adds the following new capabilities to the Meta-DENDRAL program.

1) The ability to augment the original training data to modify existing rules, without the need to reconsider the original data.

2) The ability to determine to what degree each rule has been learned, and to provide a means of using partially learned rules in a reliable manner.

3) A new strategy for forming rules from the training instances inferred by INTSUM. In contrast to the old strategy (implemented in the RULEGEN and RULEMOD programs), this new strategy is conceptually simpler, considers both positive and negative evidence of potential rules from the beginning of the search, considers alternate versions of each rule, bypasses the "coarse search" which RULEGEN conducts in forming plausible rules, and allows forming rules which focus on the most "important" or most "interesting" training instances first.

4) A more complete method for considering alternate plausible versions of each rule, so that versions of the rule with better evidential support may be found.

These capabilities are illustrated in the following sections using examples taken from the domain of mass spectroscopy. The programs which define, revise, and reason with version spaces have been implemented so that they apply as well to the similar concept learning problem in the ^{13}C NMR domain. The uses of version spaces in this domain are analogous to their use described here for the mass spectroscopy problem.

6.5.1 Modifying Existing Rules Using New Data

One feature of the version space approach is the capability to process training instances sequentially, without backtracking to reconsider either previously examined training instances or previously rejected concept descriptions. This feature is exploited in Meta-DENDRAL to overcome its earlier inability to modify existing rules using subsequent training data. Once the version space of some rule is obtained, either from the positive and negative instances associated with some rule formed by RULEGEN and RULEMOD, or directly from clusters of INTSUM suggested training instances (as described in the following section), the version space may be refined using subsequent training instances.

Because a *set* of rules (and associated versions spaces) is required to cover a reasonable portion of the training data, it is necessary to determine which positive and negative training instances should be used to update which version spaces.

All negative instances are used to update each version space since each rule (or version space) in the set should be consistent with every negative instance. Nevertheless, because of the several sources of inconsistency associated with this learning problem, it is sometimes impossible to obtain rules which match an acceptable number of positive instances without matching some negative instances. If all of the patterns in each of the multiple version spaces (see chapter 5 discussion of dealing with inconsistency) match a negative instance, it is accepted as negative evidence associated with the rule, and the version space is not altered.

In order to determine whether a given positive instance should be used to train

a given version space, the version space is used to classify the instance. If the instance is classified as a negative instance by every pattern in the version space,[1] then there is no concept description consistent with this new instance as well as past instances associated with this rule, and the instance is rejected[2]. If, on the other hand, there is some pattern in the current version space which matches the instance, then the instance is accepted as a training instance for the current version space.

The use of version spaces described above to filter possible training instances is one use of the partially learned concept represented by a version space. Although the algorithm for revising the version space in response to specified training instances obtains results independent of the order in which the instances are presented, the above procedure for screening possible positive instances is itself order dependent. Because the acceptance or rejection of a training instance depends upon the contents of the version space when that instance is encountered, an instance may be accepted if presented early in the training sequence, but rejected if presented later, when the description of the disjunct has become less ambiguous. The above method for assigning positive instances to individual rules (version spaces) is therefore, order dependent, and works best if positive instances which truly "belong" to the same disjunct are presented together early in the sequence.

Once an instance has been accepted or rejected as a training instance for a given version space, that version space is updated according to the algorithm described in section 6.4.

[1] When multiple version space boundaries are used as described in chapter 6 to deal with inconsistency, the weakest (outermost) pair of boundaries is used to classify the instance.

[2] The instance may still, of course, be used to train the version space associated with some other rule in the disjunctive set.

6.6.1.1 Some Experimental Results

Table 6.1 summarizes the results of applying this method of assigning instances and updating version spaces for a set of aromatic ester data. In this case, RULEGEN and RULEMOD were used (without the aid of version spaces) to form rules from a set of 4 aromatic ester molecules analyzed by INTSUM. A set of 3 rules was formed, whose predicted fragmentations (both evidenced and unevidenced) for the set of training molecules are summarized in the first column of table 6.1.

Version spaces were then formed for each rule by using the evidenced fragmentations associated with the rule as positive training instances for the associated version space. The original training data were then applied as described above to further modify each version space. This step resulted in a set of version spaces whose predicted fragmentations for the same set of 4 training molecules are summarized in the second column of the table. Notice that the complete examination of alternative rule versions afforded by version spaces allowed determining better versions of two of the three rules produced by RULEGEN and RULEMOD - without examining new data.

Rule	As generated by RULEGEN and RULEMOD		Version Spaces determined from same training data		Version Spaces refined using additional data	
	Predictions		Predictions		Predictions	
	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect
Rule1	6	0	6	0	14	0
Rule2	5	0	7	0	17	0
Rule3	6	2	10	2	20	5

Table 6.1: Using version spaces to revise Meta-DENDRAL rules.

The resulting version spaces were then modified by examining 4 additional training molecules and their associated spectra. Each version space in the set was refined by eliminating patterns inconsistent with new training instances. The total number of correctly and incorrectly predicted fragmentations for all 8 training molecules is summarized in the final column of the table. The processing of the additional four training molecules both reduced the number of alternate rule versions in each version space, and increased the evidential support of the remaining rule versions.

In addition to the above use of version spaces to refine rules formed by RULEGEN and RULEMOD, version spaces have been used to learn rules directly from

clusters of INTSUM inferred training instances. The following section describes the later use of version spaces.

6.5.2 Inferring Rules from INTSUM Training Instances

In order to use version spaces to infer a disjunctive set of rules directly from the INTSUM proposed training instances, it is advisable to provide the program with an initial indication of which positive instances are likely to be explained by a single disjunct, or rule. This initial estimate is necessary because the method described above for assigning instances to version spaces is based upon information already learned and summarized by the version space.

Since the rules formed by Meta-DENDRAL are rules for classes of similar compounds, the data input to the program is typically a set of molecules sharing some common substructure, or chemical skeleton. Because of this similarity among training molecules, training instances corresponding to sites in different molecules, but which occupy the same position in the skeleton common to these molecules may reasonably be clustered together and assigned to a single rule. This step is currently done by hand.

The program uses this suggested cluster of positive instances to initialize the version space for each disjunct. Instances in the cluster are considered one by one, and the version space updated accordingly. Additional training instances from the same training molecules are then considered, assigned to version spaces of alternate disjuncts as described in the previous section, and used to further refine the version

space. The use of multiple version space boundaries allows the program to recover from errors in the initial clustering of training instances, and in the screening of subsequent training instances. There is, of course, a limit on the number of errors from which the program can recover. This limit is related to the number of version space boundaries maintained (as discussed in chapter 6).

Figure 6.5 illustrates two training molecules belonging to a class of compounds called androstanes, with a common skeleton consisting of atoms 1 through 19 in each molecule. These molecules were two of thirteen provided with their associated mass spectra as training data to Meta-DENDRAL. Plausible training instances (fragmentations) suggested for the two molecules by INTSUM include the indicated fragmentations. All atoms except atom 20 in each molecule are carbon atoms. In both molecules atom 20 is an oxygen atom. The difference between these two structures is the location of the single oxygen atom.

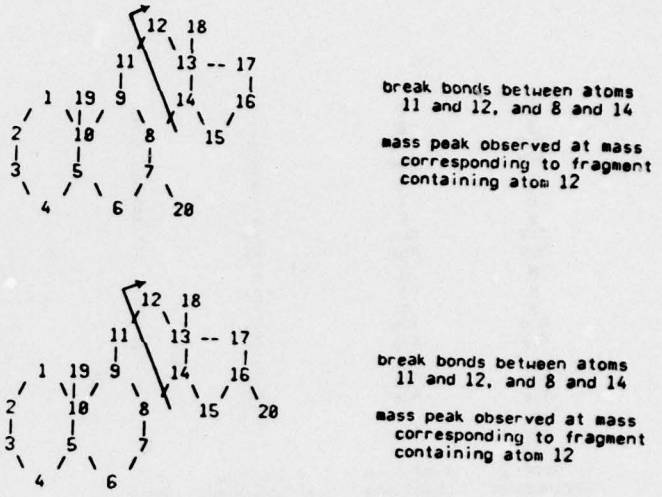


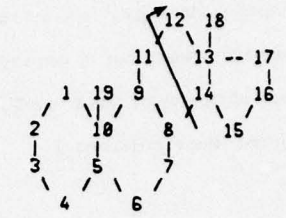
Figure 6.5 - Two training instances associated with the same version space.

A cluster of similar positive training instances, such as the two in the above figure, which contain fragmentations in several molecules which occur at the same position in the common skeleton, are suggested as instances to be covered by a single rule. Each suggested cluster provides the starting point for training the version space of a single disjunct. After these instances have been processed, the program considers all negative training instances in all molecules, refining the version space as

needed for each. In this way, a version space is formed which contains rules consistent with at least the initial cluster of positive instances, and the fewest possible negative instances.

Figure 6.6 illustrates the version space formed from training instances which exhibited the common fragmentation depicted in figure 6.5, including the two molecules shown in the figure as well as ten others out of the set of thirteen. In this example, the sizes of the boundary sets G and S were limited to a maximum of 10 patterns. The version space in figure 6.6 contains all possible rules which predict the common fragmentation which occurred in 12 out of the 13 training molecules, and which make no incorrect predictions within the 13 molecules. Only a single pattern from each boundary set is shown in the figure. There were 10 patterns in the G boundary, and 1 pattern in the S boundary at this point.

Single pattern in specific boundary set:



NODE	NODE CONSTRAINTS			
	Atom Type	Non-Hydrogen Neighbors	Hydrogen Neighbors	Unsaturated Electrons
1	C	≥2	any	any
2	C	2	2	0
3	C	≥2	any	any
4	C	≥2	any	any
5	C	3	1	0
6	C	≥2	any	any
7	C	≥2	any	any
8	C	3	1	0
9	C	3	1	0
10	C	4	0	0
11	C	≥2	any	any
12	C	≥2	any	any
13	C	4	0	0
14	C	3	1	0
15	C	2	2	0
16	C	≥2	any	any
17	C	≥2	any	any
18	C	1	3	0
19	C	1	3	0

One of 10 patterns in the general boundary set:

NODE	NODE CONSTRAINTS			
	Atom Type	Non-Hydrogen Neighbors	Hydrogen Neighbors	Unsaturated Electrons
8	any	≥2	any	any
9	any	3	any	any
11	any	≥2	any	any
12	any	≥1	any	any
14	any	≥2	any	any
15	any	≥1	any	0

Action associated with patterns in the version space:

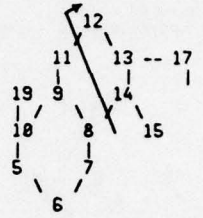
1. Break bonds between atoms 11 and 12, and between atoms 8 and 14.
2. Predict a peak corresponding to the mass of the resulting fragment which contains atom 12.

Figure 6.6 - **Meta-DENDRAL** version space.

Additional positive instances are then considered, screened, and assigned as positive instances for the version space according to the scheme described earlier using the partially learned version space. Figure 6.7 illustrates the version space obtained in this way, starting with the version space shown in figure 6.6. The

resulting version space contains rules which cover a total of 26 positive instances in the 13 training molecules - the original cluster of 12 positive instances and 14 additional positive instances chosen by the screening procedure. At this point the G boundary set contains 1 pattern, and the S boundary set contains 2 patterns. Figure 6.8 shows the predictions of rules in the final version space for 2 of the 13 training molecules.

One of 2 patterns in the specific boundary set:

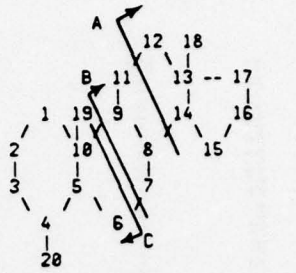


NODE	NODE CONSTRAINTS			
	Atom Type	Non-Hydrogen Neighbors	Hydrogen Neighbors	Unsaturated Electrons
5	C	≥2	any	0
6	C	≥2	any	any
7	C	≥2	any	any
8	C	≥3	any	0
9	C	3	1	0
10	C	≥3	any	any
11	C	≥2	any	any
12	C	≥2	any	any
13	C	≥3	any	0
14	C	≥3	any	0
15	C	≥1	any	0
17	C	≥2	any	any
19	any	≥1	any	any

Single pattern in the general boundary set:

NODE	NODE CONSTRAINTS			
	Atom Type	Non-Hydrogen Neighbors	Hydrogen Neighbors	Unsaturated Electrons
8	any	≥2	any	any
9	any	3	any	any
11	any	≥2	any	any
12	any	≥1	any	any
14	any	≥2	any	any
15	any	≥1	any	0

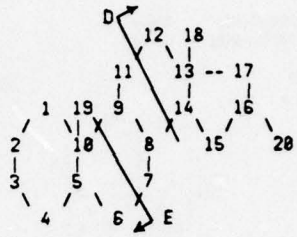
Figure 6.7 - Version space from figure 6.6 after consideration of 14 additional positive instances.



A: break bonds between atoms 11 and 12, and 8 and 14 record fragment containing node 12

B: break bonds between atoms 6 and 7, and 10 and 9 record fragment containing node 7

C: break bonds between atoms 7 and 6, and 9 and 10 record fragment containing node 6



D: break bonds between atoms 11 and 12, and 8 and 14 record fragment containing node 11

E: break bonds between atoms 7 and 6, and 9 and 10 record fragment containing node 7

Figure 6.8 - Fragmentations in training molecules predicted by the version space shown in figure 6.7.

The above procedure, beginning with reasonable clusters of positive instances derived from evidenced fragmentations in the common skeleton of several training molecules, yields rules and version spaces such as the one above. In general, these rules apply to many instances in addition to the initial cluster, and may predict several fragmentations within each training molecule, as illustrated in figure 6.8.

The above example illustrates the use of version spaces for learning rules of mass spectrometry from the plausible training instances suggested by INTSUM. In the above class of androstanes, the five most commonly evidenced skeletal breaks were used to provide five clusters of instances to initialize version spaces for five potentially different rules. The evolution of the version space corresponding to one of these instance clusters is shown above. The end result of following the above training process for each of the five instance clusters is summarized in table 6.2. The resulting version spaces contained rules which correctly predicted 77 fragmentations over the training set of 13 androstanes, with only a single incorrect prediction. Table 6.2 also summarizes the results of an earlier run of Meta-DENDRAL, using the RULEGEN and RULEMOD programs to infer rules from the INTSUM results. The five best rules obtained by RULEGEN and RULEMOD correctly predicted 87 fragmentations in the same 13 training molecules, with 5 incorrect predictions.

Rule inference program	*Predicted Fragmentations*	
	Correct	Incorrect
Version space program	77	1
RULEGEN and RULEMOD (5 best rules)	87	5

Table 6.2: Summary of predictions on training data by two sets of rules for 13 mono-keto-androstanes.

6.6 Lessons and Limitations

One conclusion to draw from experimenting with version spaces in Meta-DENDRAL, is that they provide the basis for a sound, easily characterized approach to an important part of a complex rule learning task. Here we examine the limitations of the approach illustrated by this implementation.

6.6.1 Sizes of the Boundary Sets

The most severe limitation on applying the version space approach in this domain *is the problem of efficiency.* Although the 6 androstane rules discussed above were generated using the version space approach in about a third the time which RULEGEN and RULEMOD would require, the approach is still limited by computer resource requirements. Meta-DENDRAL is a research experiment - not a production level program - and its implementation in INTERLISP is designed to allow easy restructuring of the control strategies, as opposed to efficient but fixed processing of data.

Still, the efficiency problem in the version space section of Meta-DENDRAL is directly related to the large boundary sets sometimes needed to represent version spaces. Because of the cost of dealing with these large boundary sets, options exist for heuristic pruning of the boundary sets based upon the several flexible criteria described above.

6.6.2 Need for Domain Knowledge

Syntactic methods alone are sometimes insufficient in Meta-DENDRAL to deal

with the combinatorics involved in rule learning. Because the language of chemical substructures employed by Meta-DENDRAL is redundant, knowledge of the interdependencies of the node properties (including outside knowledge concerning the valances of atoms) is needed to prevent inefficiency in representing version spaces. By introducing this knowledge, redundant rules in the version space may be eliminated without affecting the final results.

In addition, assumptions concerning the particular class of compounds under consideration may be used to prune from the version space those rules which do not fit the assumptions. Allowing more flexible ways to constrain the range of rules considered is a desirable extension for the current program. One step in this direction is to allow the user to input particular subpatterns which must not appear in a rule. This would allow, for example, prohibiting rules which break bonds between atoms with no unsaturated electrons. Such constraints could be used to prune the version space in much the same way that positive and negative training instances are currently used.

6.6.3 Learning Disjunctive Sets of Rules

The need to provide an initial cluster the INTSUM produced training instances to begin the generalization process illustrates a limit on the use of version spaces for learning large sets of disjunctive rules. Although the extended version space approach described in chapter 6 does allow learning in inconsistent situations, including learning disjunctive sets of rules, there is a limit on the number of training instances which a single disjunct must cover in order for the approach to be efficient.

Because each rule learned in Meta-DENDRAL typically covers only a small percentage (< 5%) of the positive instances, many alternate specific boundary sets must be maintained in order to learn a set of rules that cover the training data. This limitation is circumvented in Meta-DENDRAL by introducing a plausible initial cluster of positive instances to initialize each rule, or disjunct. Given this starting point, the program can use the information summarized in the version space to screen possible new instances, and succeeds in learning a useful set of rules to cover the data.

6.6.4 Need for Training Instance Selection

The use of version spaces to generate training instances to direct future learning, outlined in chapter 4, may be especially useful in the Meta-DENDRAL domain. Because the cost of obtaining training data in this domain is high, a method for suggesting highly informative new training data is attractive for economies in both obtaining and processing data. In addition, if instances could be selected to control the sizes of the version space boundary sets, heuristic pruning could be reduced, resulting in a more complete examination of the rule description language. Even if the program were able to determine which of a large library of available spectra were potentially informative, the savings in entering and processing data could be substantial.

It is likely that any useful program for suggesting new training data would make use of substantial knowledge of chemistry (e.g. stability of possible molecules, cost of synthesis) in addition to the information summarized in version spaces. This appears to be a rewarding avenue for further research in Meta-DENDRAL.

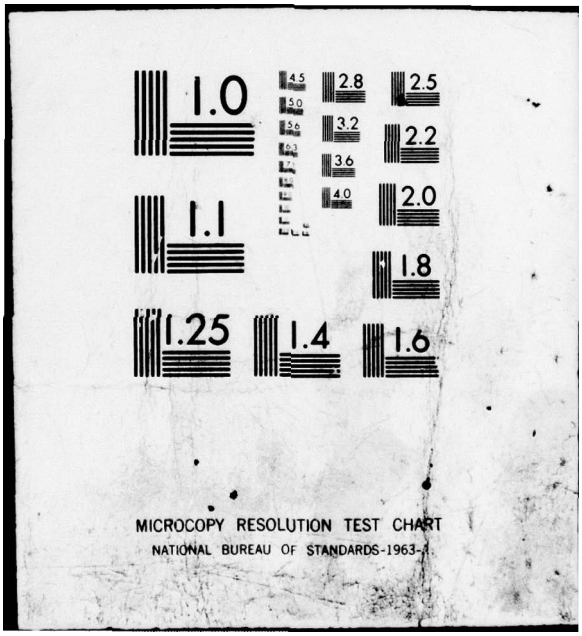
6.7 Summary

The use of version spaces in Meta-DENDRAL has been described. It has been shown that a central part of the rule learning problem in Meta-DENDRAL corresponds to a concept learning problem to which the version space approach is well suited. Version spaces have been used to add several important capabilities to Meta-DENDRAL: to extend its proficiency in examining a space of possible rules, to allow using new training data to revise existing rules, and to provide an alternative to the major rule inference programs RULEGEN and RULEMOD. Version spaces provide a more complete method than RULEMOD for considering possible revisions to RULEGEN produced rules, as demonstrated in the experiment describing the processing of the ester data. If provided with a rough initial clustering of training instances determined by INTSUM, version spaces provide an alternate strategy to RULEGEN for forming disjunctive sets of rules.

AD-A074 462 STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE F/G 9/4
VERSION SPACES: AN APPROACH TO CONCEPT LEARNING.(U)
DEC 78 T M MITCHELL DAHC15-73-C-0435
UNCLASSIFIED STAN-CS-78-711 NL

2 OF
AD
A074462

END
DATE
FILMED
10-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Chapter 7

Comparison With Alternate Approaches

Learning concept descriptions from sets of training instances is a problem which has received attention by many researchers. In this chapter the version space approach is compared to several earlier approaches to concept learning. In keeping with the view of concept learning as a search problem, the alternate approaches are grouped according to the search strategies which they employ. Comparisons are made between these classes of strategies in terms of their capabilities and efficiency.

7.1 Concept Learning as Search

The class of concept learning problems considered throughout this thesis is the following:

Concept Learning Problem:

Given: 1. A concept description language.
2. An associated pattern matcher.
3. Sets of positive and negative training instances of the target concept.

Determine: Concept descriptions within the given language which are consistent with the provided training instances.

This concept learning problem may easily be phrased as the following search problem.

Search Problem:

Given: 1. Search space of possible hypotheses.
2. Constraints on acceptable hypotheses.

Determine: Hypotheses within the search space which are consistent with the provided constraints.

The correspondence between the above search problem and the above concept learning problem is as follows: The search space of possible hypotheses is the space of concept descriptions which can be stated in the provided language. Constraints on acceptable hypotheses are the training instances of the target concept. The pattern matcher provides a procedure for applying training instances as constraints on acceptable hypotheses. Given a particular constraint (training instance), and a hypothesis (concept description), the hypothesis satisfies the constraint if either (1) the training instance is a positive instance and matches the concept description, or (2) the training instance is a negative instance and does not match the concept description.

This view of concept learning as a search problem has been held by many researchers [Amarel, 1971], [Winston, 1970], [Simon, 1973], [Hayes-Roth, 1975] and provides a useful basis for characterizing alternate strategies to concept learning. Given this perspective, there are at least as many approaches to concept learning as there are methods for examining the space of possible concept descriptions.

7.2 Model-Driven and Data-Driven Search Strategies

Two broad classes of search strategies that have been employed for concept learning problems may be called model-driven and data-driven search strategies. These terms have been used ([Buchanan, 1978], [Feigenbaum, 1977], [Stefik, 1978]) to refer to procedures for examining search spaces of possible hypotheses in several problems outside the concept learning problem considered here. In a model-directed approach, hypotheses are generated according to a predetermined model based on knowledge of the problem, and are then tested against the set of training instances (hypothesis constraints). In contrast, in a data-directed approach hypotheses are generated by considering training instances one at a time to determine which hypotheses are to be considered next. Thus, model-directed approaches tend to consider all available constraints at each step to test the methodically generated hypotheses, whereas data-driven approaches consider the data constraints one at a time to generate new hypotheses.

The candidate elimination algorithm belongs to the class of data-driven strategies: each revision to the version space of plausible hypotheses is initiated by consideration of a single training instance. Similarly, Winston's program for inferring descriptions of "blocks-world" concepts [Winston, 1975] and Hayes-Roth's interference matching algorithm [Hayes-Roth, 1975] are data-driven approaches. In each of these programs, training instances are processed *sequentially* in such a way that effects of individual instances on final results are cumulative.

Other programs, such as the RULEGEN portion of the Meta-DENDRAL program

[Buchanan, 1978] are model-driven in the sense that modifications to current hypotheses are generated systematically by a generator based on some model of the problem. In RULEGEN, all available training instances are used to prune the model-driven search at each step, but it is the predefined model - not the data - that determines which hypotheses are to be generated and considered in the next step of the search.

The data-driven strategies referred to above are well suited to sequential processing of training instances, since they make decisions by considering each training instance individually, in a quite local context. Model-directed strategies, such as RULEGEN, consider hypotheses according to a predetermined model, and are not designed to reconsider previously examined hypotheses when new data becomes available. As a result, model-driven strategies are not as well suited to the problem of revising partially learned concepts on the basis of new data.

Because data-directed approaches make decisions based upon individual training instances, they cannot easily protect against the effects of occasional errors in the training instances. Model-directed strategies can more naturally accommodate errors in the training instances because individual training instance errors are smoothed out by considering *many* training instances at once to prune the generated hypotheses.

In summary, data-directed strategies are well suited for revising partially learned concepts when new training instances become available, and for efficient processing of reliable training instances. Model-directed approaches are well suited

to problems involving unreliable training instances. These results are consistent with the conclusions of other researchers considering hypothesis formation tasks other than concept learning [Nii, 1978], [Stefik, 1978].

One interesting compromise for taking advantage of the strong points of both model-directed and data-directed strategies is found in Meta-DENDRAL. In one organization of this program, described in chapter 6, the initial stage of concept learning is accomplished by the RULEGEN program which conducts a coarse, model-driven search of the space of possible hypotheses (formed from highly unreliable training instances). A data-driven strategy (candidate elimination using version spaces) is then used to refine these rules using both the original and subsequent training instances. Thus, the first analysis of the data is conducted by a model-driven strategy, followed by a more complete, data-driven analysis of the initial and subsequent data.

Although candidate elimination is a data-directed strategy, it can be extended, as described in chapter 5, for learning in the presence of inconsistency. As a result, it has the advantages associated with data-driven strategies, but with a capability to recover from data errors.

7.3 Three Data-Directed Strategies

A range of data-directed search strategies has been employed by previous concept learning programs. Although no two programs use exactly the same strategy, it is useful to group the programs into classes whose members employ similar

strategies and therefore possess similar performance characteristics. The aim of this chapter is not to compare alternate concept learning *programs*, but rather alternate *general strategies* which existing programs implement in various ways. We consider two general search strategies which are well represented in the literature, in addition to the version space approach. Prototypical examples of each of these three strategies are described below. Characteristics of the prototypical example are then described and compared.

7.3.1 Depth-First Search - Current Best Hypothesis

Several programs in the literature (e.g., [Winston, 1970], [Waterman, 1970]) employ a depth-first search strategy for searching the space of hypotheses (concept descriptions). In this strategy, a *current best hypothesis* is determined in some way, then revised on the basis of examining subsequent training instances one at a time. If the current hypothesis is consistent with the new training instance, it is left unchanged. If the hypothesis is inconsistent with the new instance, it is revised so that it becomes consistent with the new instance as well as the old instances. Each revision to the current hypothesis corresponds to a step in a depth-first search through the space of possible hypotheses. This revision to the hypothesis involves two steps. (1) Determining a set of possible revisions which are consistent with both the current and previous training instances. (2) Selecting one such modification as the new current best hypothesis.

7.3.2 Breadth-First Search - Several Alternate Hypotheses

Several programs have been written (e.g., [Plotkin, 1970], [Hayes-Roth, 1975], [Vere, 1977]) which employ a breadth-first search strategy to examine possible concept descriptions. In contrast to the depth-first search strategy, this strategy involves maintaining a set of *several current hypotheses* corresponding to alternate concept descriptions consistent with the training instances. The above programs represent an important step in taking advantage of the general-to-specific partial ordering on the search space of concept descriptions. Each of these programs uses this partial ordering to direct the breadth-first search so that progressively more general concept descriptions are considered each time the set of current hypotheses must be modified.

Here we describe a prototypical breadth-first search from specific to general concept descriptions. The approach begins by initializing the set of current hypotheses to the most specific concept description(s) consistent with the first positive training instance. Subsequent training instances are then considered. If the instance is a positive instance, any concept description from the current set which does not match the instance is replaced by the set of least more general concept descriptions which match the instance. This set is then pruned by eliminating any new member which matches some past negative instance. If the new instance is a negative instance, concept descriptions from the current set which match the instance are deleted. This procedure determines a set of current hypotheses which correspond exactly to the *specific boundary set of the version space* associated with the observed instances. Thus, the breadth-first search strategy corresponds to

determining one of the boundary sets of the version space. This approach is therefore assured to find the set of maximally specific concept descriptions consistent with the training instances. This guarantee has been proven for particular concept description languages by Plotkin [Plotkin, 1970], Hayes-Roth [Hayes-Roth, 1974], and Vere [Vere, 1975].

7.3.3 Candidate Elimination - All Plausible Hypotheses

The candidate elimination algorithm and version spaces have been described in detail in earlier chapters. In contrast to the above search strategies, the candidate elimination algorithm represents and revises the version space associated with observed instances. Thus, it considers the set of *all plausible hypotheses* consistent with the set of observed training instances. The version space is initialized contain all concept descriptions consistent with the first positive training instance. Each subsequent training instance is then used to eliminate from the version space those candidate concept descriptions which are inconsistent with that instance.

The version space is represented by the sets of its maximally specific and maximally general concept descriptions. Each of these boundary sets of the version space traces out a breadth-first search during the course of processing training instances. The candidate elimination approach can therefore be seen as an extension to the earlier breadth-first search programs which determine the specific boundary of the version space. In addition to the set of maximally specific concept descriptions maintained in the breadth-first search strategy, a second set of maximally general concept descriptions is maintained by the candidate elimination strategy. By keeping

both of these sets, an important new kind of information is available to the learning program. The importance of this information in providing additional capabilities for the candidate elimination strategy is summarized in the next section.

7.4 Capabilities

The central concept learning task is to determine concept descriptions which are consistent with the training instances. The main difference between the depth-first search, breadth-first search, and candidate elimination strategies is the number of correct concept descriptions which they are able to determine. Depth-first search attempts to find a single acceptable concept description. The breadth-first search strategy characterized above is assured to find all maximally specific such concept descriptions. The candidate elimination approach determines every concept description consistent with the training instances.

This difference in the number of concept descriptions determined by the various approaches is significant for several reasons. By summarizing *all* plausible concept descriptions, the candidate elimination strategy avoids having to reconsider past training instances. There is no need to check which revisions to the current set of hypotheses are consistent with past data. All concept descriptions in the current version space are consistent with past data. For concept learning problems involving a large number of training instances, this is an important feature.

The information summarized in the version space boundary sets is also useful for determining to what extent the target concept has been described by the training

instances. If there are many concept descriptions in the version space, the training instances observed so far do not contain sufficient information to uniquely describe the target concept. In this case, the partially learned concept may still be used to classify new instances as either instances of the concept, not instances of the concept, or instances which cannot be reliably classified on the basis of the observed data. The procedure for classifying new instances on the basis of partially learned concepts is discussed in chapter 4.

By summarizing all concept descriptions consistent with the current set of training instances, version spaces also provide the information required to generate informative new training instances. This use of version spaces is discussed in chapter 4.

For inconsistent learning problems, in which concept descriptions consistent with every training instance do not exist, an alternate goal for the learning program is to determine concept descriptions consistent with the largest possible subset of the training instances. The candidate elimination algorithm may be extended as described in chapter 5 to determine all such concept descriptions in the presence of limited inconsistency. For more severe inconsistency, an approximate approach based upon the exact approach is available. It appears that the methods described in chapter 5 may be extended to apply as well to the breadth-first search strategy characterized above. Hayes-Roth [Hayes-Roth, 1974] has described a different extension to the breadth-first search strategy which allows dealing (in an approximate manner) with inconsistency.

7.5 Efficiency

Efficiency considerations are an important practical consideration when comparing alternate concept learning strategies. In considering the efficiency of a learning program, at least two kinds of measures should be taken into account. Processing time and storage requirements provide measures of the *computational resources* required to perform a learning task. It is important to consider as well the *human resources* required to perform the task.

7.5.1 Training Resource Requirements

For many learning tasks the chief human resource cost is in supplying training information to the program. In the Meta-DENDRAL program, for instance, the cost of supplying training information to the program includes the costs of acquiring samples of compounds (this often requires synthesizing the compounds in the laboratory) and obtaining their mass spectra. For problems such as this, human resource costs are high enough that a significant investment of computer resources is warranted for choosing well planned training instances. As we look forward to a technology which will provide less expensive, faster computers, we can expect the scale along which we measure machine efficiency to shift in favor of using more machine resources, further enhancing the relative importance of human resource costs in this tradeoff.

The importance of careful selection of training instances for efficient and reliable learning has been stressed by several writers [Winston, 1970], [Simon, 1973], [Smith, 1977], yet few learning programs take an active role in determining

their own training instances[1]. As demonstrated in chapter 4, version spaces summarize the information needed to propose informative new training instances. This capability could have a significant effect on lowering the costs of obtaining and processing training instances. To a lesser degree, the breadth-first search approach provides similar information in the set of several alternate hypotheses which it determines.

7.5.2 Computer Resource Requirements

The dependency of processing time and maximum intermediate storage requirements as a function of the number of training instances is shown in table 7.1 for the depth-first and breadth-first strategies defined above, as well as for the candidate elimination algorithm. The calculations of these values for both the breadth-first search and candidate elimination approaches are based upon the assumption that the sizes of the version space boundary sets do not grow indefinitely with the number of observed training instances. Although this assumption has not been proven formally for arbitrary concept description languages, it holds empirically for the two concept learning problems for which the candidate elimination algorithm has been implemented (the feature interval learning problem, and the Meta-DENDRAL structural concept description learning problem). This assumption is discussed further below.

[1] Exceptions include an induction program [Popplestone, 1969] which itself generates training instances whose (user supplied) classification resolves among *competing hypotheses*, and a program [Larson, 1977b] which selects "most representative" training instances from a large set of possibilities.

The dependency of processing time on number of training instances varies widely among these three approaches. Because the depth-first search described above does not take full advantage of the partial ordering on the search space, it must test each revision to the current hypothesis against all past training instances. As a result, it requires processing time proportional to the square of the number of observed training instances (denoted $O((p+n)^2)$, where p is the number of observed positive instances, and n the number of observed negative instances). In contrast, the breadth-first search strategy described above does not require reexamining positive instances. Each time that an observed positive instance alters the set of current hypotheses, those altered hypotheses are tested against previous negative instances, making the time for processing proportional to the product of p and n, assuming positive and negative instances are intermixed in the training sequence. Because the candidate elimination algorithm requires no reconsideration of previous training instances, it requires processing time which is linear with the number of observed instances.

The operations counted to determine the orders of the above search algorithms are comparisons of patterns to instances and of patterns to patterns. For simple patterns of fixed, independent features, this comparison grows linearly with the number of features. For more complex, structural patterns (such as those in the examples of chapters 2 and 6) comparison of patterns to instances may involve determining subgraph isomorphism, a known NP-complete problem [Cook, 1971]. Although the complexity of such pattern matching procedures does not affect the *relative* comparison of alternate search strategies, it is an important factor in determining the overall efficiency of any of the methods (see [Hayes-Roth, 1978]).

	Processing time	Intermediate Storage
Depth-First Search	$O((p+n)^2)$	$O(p+n)$
Breadth-First Search	$O(pxn)$	$O(n)$
Candidate Elimination	$O(p+n)$	$O(1)$

Table 7.1 Dependency of processing time and maximum storage costs on number of observed positive training instances, p, and number of observed negative training instances, n.

Table 7.1 also summarizes storage requirements of he alternate strategies as a function of the training set size. The depth-first search strategy requires storing all training instances so that future revisions to the current hypothesis may be tested against these instances. The breadth-first search strategy needs to store only negative instances for later examination because the search is organized to follow the specific-to-general partial ordering. As described previously, the candidate elimination algorithm does not require reexamining previous training instances, so its storage requirements are not proportional to the number of training instances observed.

A few additional comments on the processing time and space requirements of the candidate elimination approach are in order. The efficiency of this approach is strongly influenced by the sizes of the boundary sets which represent the version

space. The processing time required to update the version space boundary sets in response to an individual positive (negative) training instance is proportional to the square of the size of the boundary set S (G).

For the concept learning problems considered here, the sizes of the boundary sets typically behave as shown in figure 7.1. This figure illustrates general trends which hold empirically for the two concept learning programs for which the version space approach has been implemented. No formal analysis of the observed trends has been attempted.

The boundary sets are initialized to each contain a single pattern. Each set grows as training instances are observed until a plateau is reached. The set sizes vary, but tend to remain at roughly this plateau until they begin to decrease in size as the version space contains fewer and fewer concept descriptions. The maximum set size is typically larger for the general boundary than for the specific boundary.

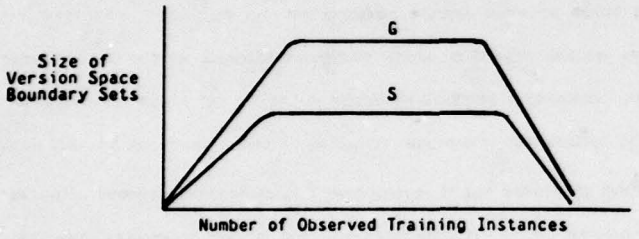


Figure 7.1 Typical Relation of Version Space Boundary Set Size to Number of Observed Training Instances.

For the feature interval learning problem, the boundary sets remained small. In

that problem, the specific boundary set can never contain more than a single pattern. The size of the general boundary set rarely exceeded 10 patterns (when multiple boundary sets are employed to deal with inconsistency, the outer boundaries may be larger as shown in table 5.1). In contrast, for the Meta-DENDRAL implementation, boundary set sizes vary with the complexity of the training molecules. For a set of aromatic esters which typically include 10 to 20 non-hydrogen atoms, the maximum boundary sizes were typically 50 patterns in the general boundary and 20 patterns in the specific boundary. For the set of androstanes illustrated in chapter 6, the maximum general boundary set size often reached 100 patterns, and the specific boundary set 30 patterns.

The sizes of the intermediate boundary sets can be strongly influenced by the order in which training instances are presented (the final sets, of course, do not depend on this ordering). Intermixing positive and negative instances seems to be the best strategy for controlling boundary set sizes. If all positive instances are considered first, the specific boundary becomes quite large. If negative instances are interspersed with the positive instances, they prune concept descriptions from this boundary, reducing its maximum size. Chapter 4 suggests guidelines for proposing training instances to control the sizes of the intermediate boundary sets.

7.6 Summary

A useful perspective on concept learning is that it is one kind of search problem in which a space of possible concept descriptions is examined subject to constraints imposed by the training instances. This perspective is useful for characterizing

various approaches to concept learning, in terms of the strategies which they employ to examine the search space.

Several classes of search strategies (concept learning strategies) were considered. In particular, three data-driven strategies for examining the space of possible concept descriptions were characterized and compared in terms of relative capabilities and efficiency. The set of concept descriptions determined by a breadth-first search strategy similar to that employed by Plotkin, Hayes-Roth, and Vere is found to correspond to the specific boundary set of the version space associated with the training instances.

The information summarized in the represented version space is useful for many tasks as described in previous chapters. The chief difference between capabilities of the version space approach and the other approaches considered may be traced to this new information available to the program.

Chapter 8

Summary and Conclusions

8.1 Results

Version spaces provide the basis for a provably correct learning procedure applicable to a broad class of concept learning problems. This candidate elimination algorithm learns concepts described in a predetermined language by examining a sequence of positive and negative training instances of the concept. The algorithm begins by representing the version space of all concept descriptions consistent with the first observed positive instance, then eliminates candidate concept descriptions from the version space as they are found to conflict with subsequent training instances. Features of the candidate elimination algorithm include:

1) All concept descriptions (within the prescribed language) which are consistent with the training instances are contained in the computed version space.

2) Backtracking is not required to reconsider either previously examined training instances or previously rejected concept descriptions. As a result, the program does not need to store past training instances.

3) Results are independent of the order in which training instances are presented.

4) Processing time is linear with the number of observed training instances.

The generality of the version space approach to concept learning has been demonstrated both theoretically and empirically. The correctness of the boundary sets representation for version spaces was formally proven for a broad range of concept description languages including any countably infinite language. The correctness of the candidate elimination algorithm for determining the version space associated with a set of training instances was also proven. The use of version spaces has been illustrated for learning concepts in three different problem domains: learning classes of simple structures made out of children's blocks, learning numerical feature value intervals, and learning rules associating chemical substructures with mass spectral peaks.

By representing the version space, a program acquires the ability to describe, and therefore reason about what can and cannot be determined about the identity of the target concept on the basis of the observed training instances. This ability to summarize the information from the training instances *in the language of concept descriptions* leads to several important capabilities. In particular, version spaces summarize information needed for solutions to the following four problems discussed initially in chapter 1.

Which Concept Descriptions Are Consistent With Observed Training Instances?

Exactly those which are contained in the version space. Programs which determine and revise a current best concept description or descriptions must examine past training instances explicitly to determine which revisions to current hypotheses

are consistent with past instances. As a result, they require processing time proportional to the square of the number of training instances. The version space approach does not reexamine training instances, and therefore requires processing time which is linear with the number of training instances (see chapter 7).

When Is a Given Concept Unambiguously Learned?

When only a single concept description remains in the version space. Furthermore, partially learned concepts can be represented and reliably used. Learning a concept corresponds to eliminating concept descriptions from the version space so that exactly those consistent with the data remain. When there is more than one concept description in the version space, the concept is not uniquely determined by the training instances. In such cases, the version space may be used to classify new instances in a reliable manner even though the identity of the concept is not completely determined. Chapter 4 describes this use of version spaces to classify instances as (1) instances of the concept, (2) not instances of the concept, or (3) instances which cannot be reliably classified without further training data.

Are the Training Instances Consistent?

If and only if the version space is not empty. Furthermore, an extension allows learning from inconsistent instances. By definition, inconsistency arises when there is no concept description which matches all positive instances and no negative instances. Such inconsistency is detected when the version space contains no concept descriptions. By generalizing the definition of version spaces and the candidate elimination algorithm, concepts can be reasonably learned in the presence

of such inconsistency. For limited inconsistency, all concept descriptions consistent with the largest consistent subsets of training instances are determined. For more severe inconsistency, an approximate method based on this exact method is presented (see chapter 6).

What Additional Training Instances Would be Informative?

Any instance which matches some, but not all, concept descriptions in the current version space. By generating such instances, whose classification will allow eliminating candidate concept descriptions from the version space, a program can propose informative instances without knowing the identity of the target concept. An optimal strategy for proposing such training instances is illustrated for the feature interval learning problem. Guidelines are also given for choosing training instances which improve processing efficiency by controlling the sizes of the version space boundary sets (see chapter 4).

8.2 Assumptions and Limitations

The method for representing and revising version spaces is defined and proven in terms independent of the language chosen for describing concepts. Implementations of the candidate elimination algorithm for the feature interval learning problem, and for the Meta-DENDRAL concept learning problem have been described. In order to use version spaces for learning concepts represented as patterns in a given concept description language, one must implement the following language-dependent procedures:

1) A pattern matching predicate for matching instances to concept descriptions.

2) A predicate "more specific than" for determining whether one concept description is more specific than another.

3) The procedure UPDATE-S which, given a positive training instance, revises the specific boundary set of a version space in a prescribed manner.

4) The procedure UPDATE-G which, given a negative training instance, revises the general boundary set of a version space in a prescribed manner.

The implementation of these procedures for the graphical language used to describe chemical substructures in Meta-DENDRAL is detailed in chapter 6. This implementation provides an example of how such procedures might be implemented for other network or structural pattern languages. These procedures have also been implemented for the feature interval learning problem described in chapter 2. Procedures similar to UPDATE-S have been implemented by others for other concept description languages [Plotkin, 1970], [Hayes-Roth, 1974], [Vere, 1975].

Although the applicability of the version space approach to concept learning problems involving a broad range of concept description languages has been proven formally, the description and processing of version spaces is more efficient for some languages than for others. Since processing time is proportional to the square of the sizes of the boundary sets, the nature of the partial ordering associated with the language strongly influences efficiency. For languages where branching in the partial ordering is deep, but not wide, the approach will be more efficient than for languages

for which the branching is more broad. For instance, in the feature interval learning problem, the boundary sets which represent version spaces are small. In the Meta-DENDRAL problem, the sizes of these sets is manageable for simple molecules in spite of the rich structural language for describing concepts. For very large molecules, the boundary sets are sometimes pruned heuristically to improve program efficiency while sacrificing completeness. For other languages, such as those which allow unlimited disjunctions of patterns, branching in the partial ordering may be too great for practical use of the boundary sets representation for version spaces.

One important topic for further work is the characterization of the sizes of version space boundary sets and the overall efficiency of the version space approach as a function of the chosen concept description language. Methods for choosing training instances to control boundary set sizes were considered in chapter 4, will be the subject of future study. A second route toward improving efficiency is to determine a more efficient scheme for representing version spaces. One interesting possibility is to represent each boundary set in terms of a single pattern common to all its elements, together with sets of allowed additional features and constraints.

Although efficiency is an important consideration for any program, proficiency is a more basic concern. Fast programs incapable of dealing with central problems will always be less useful than time consuming programs which solve these problems. Recent trends in hardware technology indicate that complex programs which require c.p.u. hours today might require c.p.u. minutes or seconds five years from now executing on less expensive computers. Progress on methods for machine learning is at a very early stage, and our time is therefore better spent developing methods

which go beyond the capabilities of current programs than in increasing the efficiency of current methods.

8.3 Future Work

The use of version spaces to propose optimal new training instances to direct concept learning was demonstrated in chapter 4 for a simple feature interval learning problem. The principle employed there generalizes to other concept description languages, but the implementation of the general method requires language-specific routines. The determination of routines to generate optimal training instances for languages of structural or network patterns is a promising avenue for further work.

Version spaces provide a useful summary of what can be known about a given concept, with respect to the concept description language, *on the basis of the training instances*. If the data are not sufficient to completely determine the concept, or if the data contain errors, then problem specific knowledge may provide additional constraints for chosing among plausible concept descriptions. The limited use of domain knowledge to eliminate concept descriptions from version spaces was considered in the Meta-DENDRAL program, and was discussed in chapter 6. A more thorough study of methods for constraining the version space on the basis of problem specific knowledge or other more general criteria (e.g., simplicity, elegance) in addition to the training data would constitute an important extension of the work reported here.

This dissertation describes a version space approach to *concept learning*, which is

used to determine the conditional part of the condition-action rules learned by Meta-DENDRAL. Further work is needed to generalize the methods discussed here to the problem of *rule learning* in which both the condition and the action must be determined. The work of Vere [Vere, 1977] and Hayes-Roth [Hayes-Roth, 1975] in defining a partial ordering for several kinds of rules and in describing programs which determine the set of maximally specific rules consistent with training instances provides a solid base for such an extension of the version space approach.

8.4 Conclusions

One process central to learning is the process of generalization. Version spaces form the basis of a powerful method for generalizing from training instances to learn concepts in a broad range of problem domains. This conclusion is supported by both theoretical and empirical results. The power of the version space approach derives from an efficient, provable method for representing and revising version spaces, and from the summary of observed training instances afforded by version spaces.

Version spaces have important uses beyond their central role in the concept learning algorithm presented here. They summarize training data information that is required to perform other tasks, such as using partially learned concepts in a reliable manner and proposing optimal new training instances.

References

The following abbreviations are used in the Reference section.

IJCAI4 *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, USSR, September 1975 (available from MIT AI Lab, 545 Technology Square, Cambridge, Mass., 02138).

IJCAI5 *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Mass., August 1977 (available from MIT AI Lab, 545 Technology Square, Cambridge, Mass., 02138).

[Amarel, 1971]
S. Amarel, Representations and modeling in problems of program formation. In *Machine Intelligence 6*, (Meltzer and Michie, eds.), University of Edinburgh Press, Edinburgh, 1971.

[Aubin, 1977]
R. Aubin, Strategies for mechanizing structural induction. *IJCAI5*, Cambridge, Mass., August 1977, pp. 363-369.

[Banerji, 1974]
R. Banerji, Learning to solve games and puzzles. In *Computer Oriented Learning Processes*, (J. C. Simon, Ed.), Noordhoff, Leyden, 1976.

[Barrow, 1972]
H. G. Barrow and R. J. Popplestone, Relational descriptions in picture processing. In *Machine Intelligence 7*, (B. Meltzer and D. Michie, Eds.), American Elsevier, New York, 1972, pp. 377-396.

[Bauer, 1975]
M. Bauer, A basis for the acquisition of procedures from protocols. *IJCAI4*, Cambridge, Mass., September, 1975, pp. 226-231.

[Bierman, 1972]
A. W. Bierman and J. A. Feldman, A survey of results in grammatical inference. In *Frontiers of Pattern Recognition*, (S. Watanabe, Ed.), Academic Press, New York, 1972, pp.31-54.

[Brown, 1977]
D. J. H. Brown, Concept learning by feature value interval abstraction. *Procedings of the Workshop on Pattern Directed Inference Systems, SIGART Newsletter 63*, 1977, 55-60.

[Brown, 1975]
J. S. Brown, et al., Steps toward a theoretical foundation for complex, knowledge based CAI. BBN Report 3135, Cambridge, MA, August 1975.

[Bruner, 1956]
J. S. Bruner, J. J. Goodnow, and G. A. Austin, *A Study of Thinking*. Wiley, New York, 1956.

[Buchanan, 1974]
B. G. Buchanan, Scientific theory formation by computer. In *Computer Oriented Learning Processes*, (J.C. Simon, Ed.), Noordhoff, Leyden, 1976.

[Buchanan, 1978]
B. G. Buchanan and T. M. Mitchell, Model-directed learning of production rules, In *Pattern-Directed Inference Systems* (D. A. Waterman and F. Hayes-Roth, Eds.), *Academic Press, New York*, 1978.

[Buchanan, 1978a]
B. G. Buchanan, et al., Models of learning systems. In *Encyclopedia of Computer Science and Technology*, 1978.

[Clancey, 1979]
W. Clancey, Tutoring rules for guiding a case model dialogue. In *International Journal of Man-Machine Studies*, (Brown and Sleeman, eds.), January, 1979.

[Cook, 1976]
C. M. Cook and A. Rosenfeld, Some experiments in grammatical inference. In *Computer Oriented Learning Processes*, (J. C. Simon, Ed.), Noordhoff, Leyden, 1976.

[Cook, 1971]
S. A. Cook, The complexity of theorem proving procedures, In *Procedings of the Third Anual ACM Symposium on Theory of Computing*, 1971, p.151-158.

[Davis, 1976]
R. Davis, Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases. STAN-CS-76-552, Stanford University, July 1976.

[Duda, 1973]
R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[Elcock, 1967]
E. W. Elcock and A. M. Murray, Experiments with a learning component in a go-moku playing program. In *Machine Intelligence 1* (Collins and Michie, Eds.), Oliver & Boyd, London, 1967, pp. 87-103.

[Evans, 1968]
T. G. Evans, A program for the solution of a class of geometric analogy intelligence test questions. In *Semantic Information Processing* (M. Minsky, Ed.), MIT Press, Cambridge, Mass., 1968, pp. 271-253.

[Feigenbaum, 1963]
E. A. Feigenbaum, The simulation of verbal learning behavior. In *Computers and Thought* (E. A. Feigenbaum and J. Feldman, Eds.), New York: McGraw-Hill, 1963, pp. 297-309.

[Feigenbaum, 1971]
E. A. Feigenbaum, B. G. Buchanan, and J. Lederberg, On generality and problem solving: a case study using the DENDRAL program. In *Machine Intelligence 6*, (B. Meltzer and D. Michie, Eds.), American Elsevier, New York, 1971, pp. 165-190.

[Feigenbaum, 1977]
E. A. Feigenbaum, The art of artificial intelligence: I. themes and case studies of knowledge engineering. *IJCAI5*, Cambridge, MA, 1977, pp. 1014-1029.

[Fikes, 1972]
R. Fikes, P. Hart, and N. J. Nilsson, Learning and executing generalized robot plans. *Artificial Intelligence*, 3, 1972, pp. 251-288.

[Findler, 1969]
N. V. Findler and W. R. McKinsie, Computer simulation of a self-preserving and learning organism. *Bull. Math. Biophysics, 31*, pp. 247-253 (1969)

[Findler, 1977]
N. V. Findler, Studies in machine cognition using the game of poker. *CACM*, *20*(4), pp. 230-245 (1977).

[Friedberg, 1958]
R. M. Friedberg, A learning machine: part 1. *IBM Journal*, *2*, pp. 2-13 (1958).

[Fu, 1974]
K. S. Fu, *Syntactic Methods in Pattern Recognition*, Academic Press, New York, 1974.

[Fu, 1975]
K. S. Fu and T. L. Booth, Grammatical inference: introduction and survey-part I. *IEEE Trans. on SMC*, *SMC-5*(1), pp. 95-111 (1975); Part II, *IEEE Trans. on SMC*, *SMC-5*(4) pp. 409-423 (1975).

[Gold, 1967]
E. M. Gold, Language identification in the limit *Information and Control*, *10*, pp. 447-474, (1967).

[Green, 1976]
C. C. Green, The design of the PSI program synthesis system. *Proceedings of the Second International Conference on Software Engineering*, San Francisco, California, October 1976, pp. 4-18.

[Griffith, 1974]
A. K. Griffith, A comparison and evaluation of three machine learning procedures as applied to the game of checkers *Artificial Intelligence 5*, pp. 137-148 (1974).

[Hardy, 1975]
S. Hardy, Synthesis of LISP functions from examples. *IJCAI4*, Cambridge, Mass. September, 1975, pp. 240-245.

[Hayes-Roth, 1974]
F. Hayes-Roth, Schematic classification problems and their solution. *Pattern Recognition*, *6*, pp. 105-113 (1974).

[Hayes-Roth, 1975]
F. Hayes-Roth and D Mostow, An automatically compilable recognition network for structured patterns. *IJCAI4*, Cambridge, MA, September 1975, pp. 356-362.

[Hayes-Roth, 1976]
F. Hayes-Roth and J. Burge, Characterizing syllables as sequences of machine-generated labelled sements of connected speech: a study in symbolic pattern learning using a conjunctive feature learning and classification system. *Procedings of 3rd Int. Joint Conf. on Pattern Recognition*. Coronado, CA, 1976, pp. 431-436.

[Hayes-Roth, 1977]
F. Hayes-Roth and J. McDermott, Knowledge acquisition from structural descriptions. *IJCAI5*, Cambridge, Mass., August 1977, pp. 246-251.

[Hayes-Roth, 1978]
F. Hayes-Roth, The role of partial and best matches in knowledge systems. In *Pattern-Directed Inference Systems* (D. A. Waterman and F. Hayes-Roth, Eds.), Academic Press, New York, 1978, pp. 557-576.

[Hedrick, 1976]
C. Hedrick, Learning production-systems from examples. *Artificial Intelligence*, *7*, pp. 21-49 (1976).

[Hunt, 1963]
E. B. Hunt and C. I. Hovland, Programming a model of human concept formation. In *Computers and Thought* (E. Feigenbaum and J. Feldman, Eds.), McGraw-Hill, New York, 1963, pp.310-325.

[Hunt, 1966]
E. B. Hunt, J. Marin, and P. T. Stone, *Experiments in Induction*. Academic Press, New York, 1966.

[Hunt, 1975]
E. B. Hunt, *Artificial Intelligence*. Academic Press, New York, 1975.

[Kanal, 1974]
L. Kanal, Patterns in Pattern Recognition: 1968-1974. *IEEE Trans on Inform. Theory, IT-20*(6), 697-722 (1974).

[Kuhn, 1970]
T. S. Kuhn, *The Structure of Scientific Revolutions*. 2nd ed., University of Chicago Press, Chicago, 1970.

[Langley, 1977]
P. W. Langley, BACON: A production system that discovers empirical laws. *IJCAI5*, Cambridge, Mass, August 1977, pp. 344-346.

[Larson, 1977]
J. Larson and R. S. Michalski, Inductive inference of VL decision rules. In *Procedings of the Workshop on Pattern Directed Inference Systems, SIGART Newsletter 63*, 1977.

[Larson, 1977b]
J. Larson, Inductive inference in the variable-valued predicate logic system VL21: methodology and computer implementation. Ph.D. thesis, Dept. of Computer Science, University of Illinois, Urbana, May, 1977.

[Lenat, 1976]
D. B. Lenat, *AM: an artificial intelligence approach to discovery in mathematics as heuristic search*, PhD Thesis, Stanford University, Stanford, California, 1976.

[Lesser, 1975]
V. R. Lesser, R. D. Fennell, L. D. Erman, and D. R. Reddy, Organization of the HEARSAY II speech understanding system. *IEEE Trans. on Acoustics, Speech, and Signal Processing, ASSP-23*(1) pp. 11-23 (1976).

[Michalski, 1973]
R. S. Michalski, AQVAL/1 - Cmputer implementation of a variable valued logic system VL1 and examples of its application to pattern recognition. *Procedings 1st International Joint Conference on Pattern Recognition*, Washington, D.C., 1973, pp. 3-17.

[Minsky, 1963]
M. Minsky, Steps Toward Artificial Intelligence. In *Computers and Thought* (E.A. Feigenbaum and J. Feldman, Eds.), McGraw-Hill, New York, 1963, pp. 406-450.

[Minsky, 1972]
M. Minsky and S. Papert, *Perceptrons*, The MIT Press, Cambridge, Mass., 1969.

[Mitchell, 1977]
T. M. Mitchell, Version Spaces: A candidate elimination approach to rule learning. *IJCAI5*, MIT, Cambridge, MA, August 1977, pp. 305-310.

[Mitchell, 1978]
T. M. Mitchell and G. M. Schwenzer, A computer program for automated, empirical 13C NMR rule formation. *Organic Magnetic Resonance, 11*, (8), 1978, pp. 378-384.

[Nii, 1978]
H. P. Nii and E. A. Feigenbaum, Rule-based understanding of signals. In *Pattern-Directed Inference Systems* (D. A. Waterman and F. Hayes-Roth, Eds.), Academic Press, New York, 1978.

[Plotkin, 1970]
G.D. Plotkin, A note on inductive generalization, *Machine Intelligence 5* (B. Meltzer and D. Michie, Eds.), Edinburgh University Press, Edinburgh, 1970, pp. 153-163.

[Plotkin, 1971]
G.D. Plotkin, A further note on inductive generalization, *Machine Intelligence 6* (B. Meltzer and D. Michie, Eds.), Edinburgh University Press, Edinburgh, 1971, pp. 101-124.

[Popplestone, 1969]
R. J. Popplestone, An Experiment in Automatic Induction. *Machine Intelligence - 5* (B. Meltzer and D. Michie, Eds.), Edinburgh University Press, 1970, pp. 204-215.

[Samuel, 1963]
A. L. Samuel, Some studies in machine learning using the game of checkers. *Computers and Thought* (E.A. Feigenbaum and J. Feldman, Eds.), McGraw-Hill, New York, 1963, pp. 71-105.

[Samuel, 1967]
A. L. Samuel, Some studies in machine learning using the game of checkers II - recent progress. *IBM Journal of Research and Development, 11*(6), 601-617, (1967).

[Shortliffe, 1976]
E. H. Shortliffe, *Computer Based Medical Consultations: MYCIN*, American Elsevier, New York, 1976.

[Simon, 1973]
H. A. Simon and G. Lea, Problem solving and rule induction: a unified view. *Knowledge and Cognition* (L.W. Gregg, Ed.), Lawrence Erlbaum Associates, Potomac, Maryland, 1974, pp. 105-127.

[Sleeman, 1979]
D. H. Sleeman and R. J. Hendley, ACE: a system which analyses complex explanations. In *International Journal of Man-Machine Studies*, (Brown and Sleeman, eds.), January, 1979.

[Smith, 1977]
R. G. Smith, et al., A model for learning systems. *IJCAI5*, Cambridge, MA, 1977, pp. 338-343.

[Solomonoff, 1977]
R. Solomonoff, Inductive inference theory - a unified approach to problems in pattern recognition and artificial intelligence. *IJCAI4*, MIT, Cambridge, Mass., 1975, Vol.1:274-280.

[Soloway, 1977]
E. M. Soloway and E. M. Riseman, Levels of pattern description in learning. *IJCAI5*, MIT, Cambridge, MA, 1977, pp. 801-811.

[Soloway, 1978]
E. M. Soloway and E. M. Riseman, Knowledge-directed learning. *Pattern-Directed Inference Systems* (D.A. Waterman and F. Hayes-Roth, Eds.), Academic Press, New York, 1978.

[Stefik, 1978]
M. J. Stefik, Inferring DNA structures from segmentation data. *Artificial Intelligence, 11*, August, 1978.

[Sussman, 1973]
G. J. Sussman, A Computational Model of Skill Acquisition. MIT AI-TR-297, August 1973.

[Uhr, 1963]
L. Uhr and C. Vossler, A pattern-recognition program that generates, evaluates, and adjusts its own operators. *Computers and Thought* (E.A. Feigenbaum and J. Feldman, Eds.), McGraw-Hill, New York, 1963, pp. 251-268.

[Vere, 1975]
S. A. Vere, Induction of concepts in the predicate calculus. *IJCAI4*, Tbilisi, USSR, 1975, pp. 281-287.

[Vere, 1977]
S. A. Vere, Induction of relational productions in the presence of background information. *IJCAI5*, Cambridge, MA, 1977, pp. 349-355.

[Vere, 1978]
S. A. Vere, Inductive Learning of Relational Productions. *Pattern-Directed Inference Systems* (D.A. Waterman and F. Hayes-Roth, Eds.), Academic Press, New York, 1978.

[Waterman, 1970]
D. A. Waterman, Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, 1(1,2), pp. 121-170 (1970).

[Waterman, 1975]
D. A. Waterman, Adaptive production systems. *IJCAI4*, MIT, Cambridge, Mass., 1975, pp.296-303.

[Winston, 1970]
P. H. Winston, Learning structural descriptions from examples, MIT AI-TR-231, September 1970.

[Winston, 1975]
P. H. Winston, (Ed.), *The Psychology of Computer Vision*, McGraw-Hill, New York, 1975.