CHANGE OF REPRESENTATION IN MACHINE LEARNING, AND AN APPLICATION TO PROTEIN STRUCTURE PREDICTION

BY

THOMAS RICHARD IOERGER

B.S., The Pennsylvania State University, 1989
M.S., University of Illinois, 1992

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1996

Urbana, Illinois

CHANGE OF REPRESENTATION IN MACHINE LEARNING, AND AN APPLICATION TO PROTEIN STRUCTURE PREDICTION

Thomas Richard Ioerger, Ph.D.
Department of Computer Science
University Of Illinois at Urbana-Champaign, 1996
Larry A. Rendell, Advisor

While many excellent induction algorithms are known for making predictions from databases in well-studied domains, learning systems still perform poorly in many difficult real-world domains, such as weather prediction or financial risk analysis. Two characteristics of real-world domains are inadequately addressed by current machine learning research. First, the difficulty in these domains is often caused by a low-level representation, which necessitates shifting to a higher-level representation. But the space of possible representations is very large, so we need intelligent methods for finding higher-level representations. Second, background knowledge is almost always available in real-world domains, which we would like to take advantage of to increase predictive accuracy. However, known roles for domain knowledge in machine learning are often inflexible, requiring the use of a specific induction algorithm or being sensitive to incorrectness or incompleteness in the knowledge.

We propose a general framework for change-of-representation based on searching for alternative representations to improve the accuracy of an underlying induction algorithm. Representations are selected as candidates by querying a strategy component, which relies on domain knowledge to suggest which alternatives to search. An evaluation component then compares these representations by applying each representation to a set of examples and running the induction algorithm on the transformed examples to empirically determine the effect of the change on accuracy. This approach provides solutions to the two characteristic problems of learning in real-world domains. First, domain knowledge is used as a heuristic to guide the search for alternative representations, enabling more intelligent decisions during change-of-representation. Second, the framework provides a flexible role for knowledge that can be used with any learning algorithm and is tolerant of uncertainty. An implementation of this framework could be used

as an interface between a human expert and a learning program in which: 1) the human uses background knowledge to generate and prioritize alternative representations, and 2) the system empirically evaluates these to discover the best change for improving accuracy.

We apply our framework for change-of-representation to the difficult, real-world domain of protein tertiary (3D) structure prediction. The best computational method to date for determining the structure of a protein from its amino acid sequence is homology modeling, which is based on sequence alignments with a protein database. Homology modeling can fail in cases where the sequence similarity is low between proteins with similar structures. However, the physical and chemical properties of amino acids are believed to relevant to protein structure. Using an instantiation of our framework, we incorporate this domain knowledge to suggest ways to change the representation of amino acid sequences. Efficient search procedures are derived from the knowledge that lead to the discovery of representations that improve the ability to predict protein structures by homology modeling.

dedicated to my wife Amy

Acknowledgements

I would like to thank Larry Rendell for being my thesis advisor. Larry has always had high standards for research, but also sees the human side of science. I have encountered many questions during my graduate years, and Larry has had the wisdom to point me in the right direction, yet allow me to find my own answers.

I would also like to thank the other faculty members who served on my committee. Each has had a significant and positive influence on this research. Jerry DeJong encouraged me to explore the formal role of knowledge in my framework. Caroline Hayes prompted me to emphasize the human-computer interaction in my research. And Shankar Subramaniam urged me to apply my research to a real-world problem, and helped me understand the science involved in protein structure prediction.

Finally, I would like to acknowledge the other graduate students in the Inductive Learning group and the Computational Biology group at the University of Illinois who have helped guide this research through many discussions and critiques of earlier work. Special thanks goes to Dan Oblinger, who has always been interested in working cooperatively to find the truth in all things and understand the world.

Table of Contents

Chapter 1

Introduction

Many good induction algorithms have been developed (Michalski, 1983; Rumelhart *et al.*, 1986; Cheeseman *et al.*, 1988; Quinlan, 1993; Aha *et al.*, 1991). The efficacy of these algorithms has been demonstrated on a wide variety of benchmark domains (Weiss and Kapouleas, 1989). However, current machine learning techniques are inadequate for learning in more difficult real-world domains. Whereas benchmark domains have usually been studied over a extended period of time (e.g. the iris database, Fisher, 1936), researchers often lack a good understanding of phenomena in a real-world domain, and machine learning itself may be being used as a tool to evolve such a theory. Examples of such domains include weather prediction (Packard, 1989), financial risk analysis (Ragavan *et al.*, 1993), and drug design (King *et al.*, 1995). In real-world domains, huge databases may be available, but simply applying one of the standard induction algorithms often produces only low predictive accuracy.

Several potential causes of learning difficulty in real-world domains have been studied. The attributes or class labels may be corrupted by noise (Quinlan, 1986). Critical information may be not be encoded in the data (intrinsic inaccuracy; Rendell and Seshu, 1990). And lack of data may also inhibit learning, which is a side-effect of concept-class complexity (Valiant, 1984).

In this thesis, we focus on another common cause of learning difficulty in real-world domains: low-level representation. A low-level representation refers to the description of examples in terms of many relatively insignificant attributes, none of which is highly predictive of the target concept (though, taken together, they are believed to be complete enough to identify the concept). Low-level representations are common in real-world domains because the domain itself may not understood well enough to know how to express the data in a few highly relevant features (effectively compressing the data; Saxena, 1991). In contrast, the benchmark domains discussed above have already had high-level features extracted which have a strong relationship to the target concept and make patterns more obvious (Holte, 1993).

Low-level representations inhibit learning because they interfere with several biases that are pervasive among known induction algorithms. Low-level representations may cause phenomena such as concept dispersion or feature interaction, which interfere respectively with the similarity bias and the greedy bias (Rendell and Seshu, 1990). Because most algorithms use these fundamental biases in some form, they may all perform uniformly poorly in a real-world domain (Ragavan *et al.*, 1993). Thus it may be necessary to change the representation itself to improve learning for such problems. The task of finding higher-level representations seems easier, or at least more automatable, than inventing a novel induction algorithm with a radically different bias. Anecdotal evidence for the advantage gained by using change-of-representation in machine learning comes from studies in domains such as checkers (Flann and Dietterich, 1986) and recognition of promoters in DNA sequences (Hirsch and Japkowicz, 1994). Research on computational learning theory also lends theoretical support by linking the difficulty of learning a concept to its concept class, which can be manipulated by changing representations (Haussler, 1988).

Several automated methods for changing representation in machine learning have been proposed (Schlimmer, 1987; Matheus, 1989; Pagallo and Haussler, 1990; Ragavan and Rendell, 1993). These methods have often been called constructive induction (Michalski, 1983), feature extraction (Devijver and Kittler, 1982), or dynamic bias (Utgoff, 1986), though related methods have been proposed in other fields such as statistics and pattern recognition. However, a ubiquitous problem that all of these techniques must deal with in some way is the *complexity* of change-of-representation. In general, there are many possible ways to change representations. For example, with feature construction, a given feature vector can usually be extended in an exponential number of ways. Most constructive induction algorithms use some constraint implicit in the algorithm for restricting the changes of representation that will be considered. But intelligent and efficient discovery of higher-level representations in many domains still remains a challenge.

Another common characteristic of real-world domains that is inadequately addressed by current machine learning techniques is domain knowledge. Learning almost always occurs within some context (e.g. scientific) that can supply various kinds of information relevant to the target concept. Knowledge can be quite heterogeneous, taking on a wide variety of forms, including symmetries and invariants, relevant scales and ranges, intermediate concepts, critical cases, causal mechanisms, etc. Most learning algorithms are not designed to make use of domain knowledge, instead attempting to construct generalizations from a set of examples alone. However, in real-world domains, there is often a great incentive to exploit any available background knowledge that might give some increase in predictive accuracy, particularly when data is limited.

Several methods are known for incorporating domain knowledge into learning. PROTOS uses knowledge to help decide the best way to weight attributes for indexing cases (Porter *et al.*, 1990). Explanation-based learning uses knowledge to explain the classification of examples, thereby indicating which features are relevant to the classification and which may be dropped from the description to generalize it (DeJong and Mooney, 1986). ILP uses knowledge to generate new terms for covering examples by the process of inverse resolution (Muggleton and Buntine, 1988). KBANN uses knowledge to define the topology of a neural network (Towell *et al.*, 1990). MIRO uses knowledge to filter the features generated by a feature construction algorithm (Drastal *et al.*, 1989). Abu-Mostafa (1994) presents a technique for using knowledge to generate artificial examples to illustrate some property of the target concept. Knowledge may also be incorporated into learning in the form of analogies, such as through determinations (Davies and Russell, 1987), or by mapping the structure of one concept onto another when they are believed to be similar (Falkenhainer *et al.*, 1989). More generally, knowledge can be considered an aspect of bias, which is anything used to select hypotheses during learning, other than the data itself (Mitchell, 1980). Russell and Grosof (1987) show how inductive biases are equivalent to non-monotonic beliefs.

Despite this variety of methods for incorporating knowledge into learning, there are three potential problems. First, knowledge in real-world domains can be very heterogeneous, and it may not be clear how to fit it into one of the known roles. So we are always looking for new roles for knowledge to play. Second, some of these roles require the use of a specific induction algorithm; for example, KBANN works only with neural nets. We want any new roles for knowledge to be more general, so that they will work with a broad class of induction algorithms, giving an expert more freedom to make choices appropriate for each domain. Finally, some of these roles for knowledge are sensitive to the correctness of the knowledge. For example, the original EBL method was criticized for being unable to learn (correctly) from an domain theory that is incomplete, incorrect, or inconsistent (Wilkins, 1990). But knowledge in many real-world domains is uncertain, and we want any new roles for knowledge to be flexible enough to tolerate this uncertainty.

So learning in real-world domains poses an additional set of challenges that need to be addressed more fully by machine learning research. Accuracy in real-world domains is often limited by a low-level representation, and finding higher-level representations can be difficult. Also, real-world domains often have extra background knowledge which is potentially useful, but which may be difficult to fit into one of the known roles, or may be uncertain. Solutions to these pragmatic issues must be found before the utility of machine learning can be extended to more difficult domains.

1.1 A Search-Based Framework for Change-of-Representation

In this thesis, we propose a new approach to change-of-representation that integrates solutions to both of these problems—how to find higher-level representations, and how to use domain knowledge—at the same time. Our approach is based on treating change-of-representation as a *search problem*. Given an induction algorithm whose accuracy we are trying to improve, along with a set of examples in some initial representation, we conduct a search through a space of alternative representations to find one that significantly improves the accuracy of the learner on the data. The benefit of describing change-of-representation in terms of search is that it exposes a well-understood role for knowledge to play in learning in the form of a *search heuristic*. Knowledge is used to decide which representations are worth considering, which both helps reduce the complexity of change-of-representation and provides a flexible mechanism for incorporating domain knowledge.

The general idea of thinking of change-of-representation as a search problem is not new. Models of learning in which a "critic" module incrementally refines a problem solver to optimize its performance foreshadow this idea (Simon and Lea, 1973; Buchanan *et al.*, 1978). Searching representation space is most directly suggested by various proposals for layered learning (Rendell, 1985; Gordon and DesJardins, 1995). At one level, induction itself is conceived of as searching through a space of hypotheses (Mitchell, 1982). Then, at a higher level, a search is conducted through the space of representations, each of which generates its own hypothesis space, to find a representation which permits a good hypothesis to be found. This approach can also be thought of as a form of "dynamic bias," if the representation of examples is thought of as an aspect of the bias the learner exploits during generalization (Tcheng *et al.*, 1989).

This approach is distinct from search through "feature space," which has been proposed to understand feature construction techniques (Donoho and Rendell, 1996). Our framework treats representations as a whole, rather than searching for individual features that facilitate learning. This approach generalizes more easily to domains not based on fixed-length feature vectors, which will be important in our application to protein structure prediction. This notion of change-of-representation is also distinct from methods for shifting the hypothesis language (Utgoff, 1986). While the ideas are related, hypothesis language refers to the internal representation used by an induction algorithm, whereas our notion of representation refers explicitly to the external encoding of examples.

We will present a framework for change-of-representation based on search that formalizes many of the accompanying details behind this intuitive concept. Our framework consists of three components: a search space component, an evaluation component, and a strategy component (see Figure 1.1). To describe the search space, we must provide a formal definition of what we mean by "representations," which constitute nodes in the space. We will define representations as a mapping of domain objects into (structured) symbols in some language (i.e. descriptions). The motivation behind this abstract definition is that we want to avoid the usual definition of representations as a "list of features," in order to cover more diverse domains, including protein structure prediction. To measure our progress through the search space, we will use an evaluation mechanism that empirically measures the effect of a change in representation on predictive accuracy of the algorithm when it is trained and tested on data transformed into the new representation.

The third component–the strategy component–is queried to determine the order in which nodes in the search space are explored. This is exactly the point at which we can take advantage of domain knowledge. Knowledge can suggest the start node for the search, and knowledge can

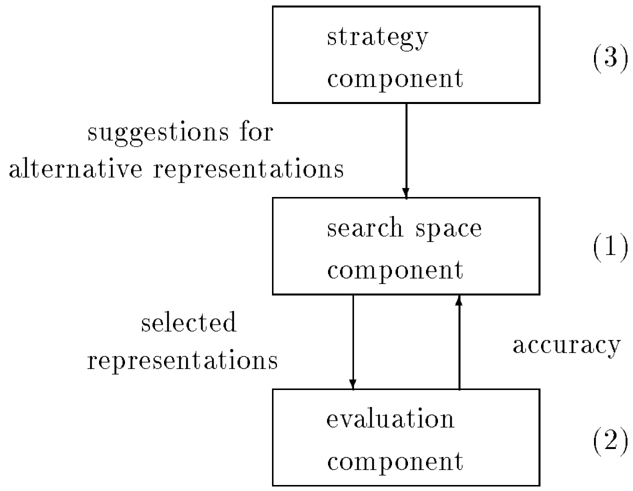

Figure 1.1: Components of the proposed framework for change-of-representation. The middle component (1) consists of nodes, which are alternative representations. The bottom component (2) is used to evaluate individual representations, which is done by running the underlying induction algorithm on some re-represented examples to determine the effect on accuracy. The top component (3) drives the search by using knowledge to help decide which representations to explore.

delimit the boundaries of the search by explicitly restricting possible changes to a certain class of syntactic manipulations (e.g. picking an appropriate feature construction operator). But knowledge can also exercise much finer control by stating specific preferences for which alternative representation is next most likely to improve learning, given the current state of the search. We will give a formal semantics for this role for knowledge in terms of constraints on extensions of partial functions that encode the search state as a mapping between representations and accuracy scores.

This third component illustrates the significance of our search-based framework: it provides a systematic role for knowledge to play in the form of a search heuristic, which is a classical notion in AI (Newell and Simon, 1972). Change-of-representation is difficult, and difficult problems are often described as having large search spaces. Knowledge can be used to help solve difficult problems by guiding the search toward nodes that meet the goal criterion more efficiently than random exploration (enumeration). In the case of change-of-representation, knowledge helps to reduce complexity by steering the search into regions of the larger space of possible alternatives that are more likely to contain improved representations. Previous methods for automated change-of-representation all have some way of dealing with complexity, but this is usually implicitly built into the algorithm. For example, FRINGE exploits the replication problem in decision trees to decide which features to combine (Pagallo and Haussler, 1990). Our framework makes these decisions about change-of-representation more explicit and *knowledge-intensive*.

This role for knowledge in learning is a fairly flexible. The framework is independent of the induction algorithm, and can be used to improve the accuracy of any algorithm that can be trained and tested on a set of examples. This is especially important in real-world domains because it allows the domain expert to use whatever induction algorithm is thought to have an appropriate bias. This role for knowledge is also tolerant of uncertainty, in the sense that incorrect knowledge might steer the search off into an unproductive part of the search space, but this merely wastes time, without actually decreasing accuracy.

1.2 Criteria for Applying the Framework

Our framework for change-of-representation can be applied to a variety of domains. In this section, we discuss several criteria that can be used to help determine whether the framework will be applicable to a given domain. The first criterion is that the domain has to be *difficult for known learning algorithms*. Furthermore, there should be reasonable belief that this difficulty is a result of a low-level representation, rather than phenomena such as noise or intrinsic inaccuracy. If examples were already described in a high-level representation, then one of the standard induction algorithms might be sufficient for learning, and there would be little room for improvement. Applying the framework makes sense only in difficult domains, and for example will not be of much use on many of the well-studied benchmark problems.

A second criterion for applying our framework is that *some domain knowledge must be available*. Change-of-representation in general is intractable, and the essence of our proposal is to take advantage of the advice of an expert to determine which alternative representations to consider. In addition, the knowledge *must be related to the representation* to be relevant. In particular, the functional formalism we will give for encoding domain knowledge in the strategy component requires that the knowledge be stated in a way that can be used to suggest improvements to representations. It may not be immediately obvious how arbitrary knowledge in some domain is related to representation. However, in many real-world domains, there is often a fairly intuitive connection between knowledge and representation based on the understanding of the meaning of the features, their relevance, and relationships among them. Other studies have focused on categorizing ways of expressing knowledge so that it is relevant to representation (Donoho and Rendell, 1996). The present research has focussed on mechanisms for using domain knowledge once it has been adapted in this way by the domain expert.

The third criterion, which gives the most insight into the utility of the framework, is that the domain knowledge should be *a little bit uncertain*, and these uncertainties should be parameterizable. If the domain knowledge were completely certain and strong, then it could be used to define a high-level representation directly, and there would be no reason to perform any search. However, search, which is central to our framework, becomes important when a domain expert holds some general belief, though he or she is unable to commit to more specific details. Such knowledge might be translated into a suggestion for the form of an alternative representation, without identifying precisely which representation in the resulting class is expected to be best for learning. If the expert can express what is uncertain about his or her knowledge in the form of a set of parameters (e.g. hypotheses), then search can be seen as essentially optimizing over this parameter space. Thus our framework is not only tolerant of uncertainty, but requires it, in some degree, to be useful. Knowledge of such intermediate strength is very common in real-world domains.

To illustrate this requirement for uncertain knowledge, consider how our framework might be applied to a domain such as weather prediction. Meteorological data are often given in large databases of low-level attributes, typically distributed over space and time. Measurements such as temperature, barometric pressure, humidity, wind speed, precipitation, etc. are sampled at various weather stations for a number of preceding days. Such a representation is difficult for standard induction algorithms because no single observation is going to have a strong predictive relationship with the overall weather pattern on the final day.

However, background knowledge can be used to suggest ways of extending the representation to make certain relevant pieces of information more explicit. For example, the low-level attributes can be combined in various ways to create summary variables that reflect patterns that are believed to be useful for predicting the weather. One possibility is to compute the average precipitation observed at several weather stations in the surrounding area. Precipitation is known to be a "regional" phenomenon, such that, at any given place and time, it either is or is not raining, but the probability of this event is correlated among nearby sites. However, it is important to choose the right radius for including stations in this average, since too large a radius will include measurements from sites so distant that their weather can be qualitatively different, which would dilute the informativeness of the average. More specific knowledge of local geographic features such as lakes or mountains could be used to recommend particular stations to include or exclude from the average, based on the expected influence of these features. Thus, knowledge generates a space of alternative representations, each of which appends a new "average precipitation" variable that is computed over some different set of neighboring weather stations, and our framework would then be used to search this space.

A similar example of change-of-representation in this weather prediction domain would be the construction of temporal averages for recognizing significant changes in important variables. For example, background knowledge suggests that a recent increase in wind speed or drop in barometric pressure might indicate that the weather will be changing. To detect such changes, the observations for one of these variables at some site must be averaged over a window of recent days. However, the size of this window should not be too large, since unrelated weather patterns in the distant past could unduly influence this computation. So the domain knowledge in this case also generates a space of alternatives because the form of the high-level attribute (another average) is known, while the best parameter (size of window) is not known. Search could be used to determine whether it is most appropriate to average a variable like barometric pressure over the preceding two days, or three days, or four days etc.

Many other real-world domains fit the criteria for applying our framework. For example, in a medical domain, a physician might want to discretize the body temperature of patients into low, medium, and high ranges, since fevers (high temp.) often indicate the presence of an infection. However, the physician might not know the best cutoff for categorizing a temperature as "high," and might want to use a simple search procedure to find a cutoff that is optimal for predicting the disease being studied. In economic forecasting, masses of data such as national commerce, finance, and employment figures are collected for making predictions. A domain expert can use background knowledge about likely correlations and causal effects to guess special combinations of variables (e.g. "indicators") that might reveal underlying patterns. Yet there is often uncertainty about which of these high-level features is most useful for prediction, and search could be used to evaluate and compare these extensions to the data. In general, our framework helps adapt standard learning methods to more difficult domains where experts have a partial theory that is still evolving; there must be enough knowledge to suggest good changes of representation, yet the theory should contain some hypotheses that are subject to testing.

1.3 Application to Protein Structure Prediction

To demonstrate our framework, we will apply it to a particular real-world problem: protein structure prediction. Our framework has enabled us to achieve significant results in this difficult but important domain (Ioerger *et al.*, 1995). Proteins are complex molecules in cells that serve many diverse biological functions (Zubay, 1983). Proteins are composed of linear chains of smaller compounds called amino acids, of which there are twenty. The DNA in a cell encodes a unique sequence of amino acids for each protein. Soon after a protein is synthesized by polymerizing (concatenating) its amino acids, it folds into unique three-dimensional shape that is essential for its function.

Biologists want to know the structures of proteins, since this gives information about their function, enzymatic mechanism, inhibition, etc. Unfortunately, it is exceedingly difficult to determine the structure of a protein using laboratory methods like X-ray crystallography, sometimes taking years of work. However, it is considerably easier to determine the amino acid sequence of a protein, such as through DNA sequencing, and it has been shown that the sequence alone is sufficient to determine the structure (White, 1961). Thus one of the grand challenges of biology is to compute the structure of a protein from its amino acid sequence.

The mapping between sequence and structure turns out to be very difficult to compute. For example, one approach, simulation of folding by energy minimization, is known to be intractable for computing structures *de novo* (from a random initial configuration) (McCammon and Harvey, 1987). Rather than trying to compute this mapping from first principles (electrostatics, Newtonian mechanics, etc.), another approach is to try to learn the relationship between sequence and structure from examples in protein databases. Protein databases, such as the Brookhaven PDB (Bernstein *et al.*, 1977), hold hundreds of entries for proteins whose structures have already been solved. The goal is to use these examples to make predictions about the structures of other proteins.

One widely used method for predicting the structures of new proteins using protein databases is called *homology modeling* (Blundell *et al.*, 1987). Homology modeling refers to the use of a sequence alignment algorithm (which inserts gaps to maximize local matches of amino acids) to detect similarities among proteins (Needleman and Wunsch, 1970). Homology modeling is based on the observation that, whenever two proteins have similar sequences (e.g. $> 30\%$ identity), they always have similar structures (Sander and Schneider, 1991). Structural similarity has a technical definition, called RMSD, that is based on measuring the deviation of corresponding atoms in the protein backbone when they are optimally super-imposed (Levitt and Chothia, 1976). However, Richardson (1981) observed that protein structures tend to fall into clusters of shapes, which she called *folds*, and a simpler expression of structural similarity is that two proteins belong to the same fold class. So homology modeling actually predicts the approximate structure of a protein by identifying its fold based on sequence similarity to a known protein, which is sufficiently informative for most purposes.

The problem with homology modeling is that, while proteins with similar sequences always have similar structures, the converse does not always hold true. Occasionally, two protein will have similar structures but have apparently dissimilar sequences. For example, there are at least fourteen proteins that have the beta-barrel shape but are all pairwise distinct (non-homologous) in sequence (Chothia, 1988). This problem, which we call the "hidden-homology problem," might cause homology modeling to make false negative predictions because, even though there might exist an entry in the protein database that is structurally similar to the protein being queried, we might fail to recognize the relationship based on sequence similarity alone.

Our goal is to improve the predictive ability of homology modeling using our framework for change-of-representation. Homology modeling is equivalent to a nearest-neighbor learning algorithm (Aha *et al.*, 1991), with sequence alignment scores playing the role of a sophisticated similarity metric. Note that examples in this domain (the amino acid sequences) are initially represented as variable-length strings over a twenty-letter alphabet, which is quite different from the fixed-length feature-vector-based domains typically studied in machine learning. There are some other methods for learning sequences (Fu, 1982; Dietterich and Michalski, 1986; Brown *et al.*, 1993), but they do not have the right bias for generalizing protein sequences. So it is convenient that our framework allows us to improve homology modeling as a domain-specific method.

To apply our framework, we need some domain knowledge, and for this purpose we can make use of biophysicists' knowledge of the chemical and physical properties of the amino acids (Dayhoff *et al.*, 1972). Based on the unique side-chain structure of each amino acid they can be described as large or small, flexible or rigid, polar or hydrophobic, aromatic, hydroxylated, charged, etc. (Taylor, 1986). It is these properties that determine what role each amino acid plays in a protein, and hence what other amino acids with similar properties can be substituted for it and fulfill the same role.

One way we can use this knowledge to suggest alternative representations is by *re-naming* the symbols in amino acid sequences. For example, given a set of amino acid property definitions, we can systematically replace the amino acid identities in all of our sequences with the names of properties possessed by those amino acids. This change-of-representation could facilitate homology modeling by making more information explicit that is relevant to protein structure. A sequence alignment algorithm could make more intelligent decisions about local matches by taking into consideration the properties of the amino acids. These new matches might increase the apparent homology between proteins with similar structures but low sequence identity, and ultimately increase the ability of sequence alignments to detect structurally-related proteins.

The problem with this approach is that biophysicists are uncertain about which properties are most important, or even what the best definitions are for them (Dayhoff *et al.*, 1972; Taylor, 1986). This is where we can take advantage of the search aspect of our framework. If a domain expert can hypothesize several alternative definitions for a particular amino acid property, and these result in slightly different changes of representation, then their effect on the predictiveness of homology modeling can be evaluated independently (using some example protein sequences) to decide which definition is best. Similarly, if the expert can propose a set of competing properties, search can be used to rank them by importance, again by evaluating their effect on homology modeling when used to change the representation of amino acid sequences. Finally, if the expert suspects that the utility of some properties might be context-dependent, where a context is defined as a subset of sites within proteins, then search can be used to determine in which context a given property is most relevant (Overington *et al.*, 1992). We will present the results of several experiments using an implementation of our framework that searches for alternative representations of protein sequences, guided by knowledge of amino acid properties, and ultimately improves the ability of homology modeling to predict protein structures.

1.4 Extension to Interactive Learning

One consequence of our framework is that it suggests a way that the knowledge input by the user might be evaluated. Given the semantic connection that our framework establishes between domain knowledge and suggestions for alternative representations, the empirical evaluation of representations can be used to critique the knowledge via feedback. In particular, if the knowledge in the strategy component suggests two representations whose improvements are expected to be in some particular order, and the actual effects of these alternative representations on the accuracy of the learner are subsequently found to be in the opposite order, then this creates an opportunity for negative feedback. Our framework allows us to generate explanations (in terms of domain knowledge) for the expected ordering of the two representations, and, with the help of a credit-assignment algorithm (Minsky, 1963), this feedback can be propagated back through the explanation to critique individual pieces of knowledge (i.e. rules). Our framework provides a foundation for assessing the correctness of knowledge through its utility in finding higher-level representations

This kind of feedback can enable a more interactive form of learning where an expert encodes some knowledge, the system does some search, the expert examines the results and revises the theory, and the process iterates. This is particularly useful in conjunction with the criterion that the domain knowledge should be somewhat uncertain to allow the search aspect of our framework to be useful. We expect that an expert who uses an implementation of this framework in some other domain will interact with the system by postulating, testing, and resolving various hypotheses about their domain. Such interaction might be crucial to the success of learning in real-world domains, since the expert's understanding of the domain is often evolving itself. Furthermore, such feedback can lead to higher-level results, where, in addition to learning the target concept, the expert may improve his or her understanding of the domain. In our experiments in the protein structure prediction domain, we used a specialized method for summarizing the results of search to produce some revisions of the knowledge about the properties of the amino acids and their role in protein structure. This analysis provided several high-level results about the amino acids and the roles they play in protein structures.

1.5 Contributions

At an abstract level, our search-based framework for change-of-representation makes several contributions to machine learning:

- It offers an improved method for change-of-representation by using knowledge to reduce complexity, thus allowing a learning system to make more intelligent decisions. The framework constitutes a knowledge-intensive method for change-of-representation, which may be needed to extend the utility of machine learning to more difficult real-world problems.

- Our framework formalizes the details behind an intuitive approach so that it can be applied systematically to a wide variety of domains, even those where examples are not described in the form of fixed-length feature vectors. The framework captures change-of-representation at a deep level as a fundamental principle in machine learning.

- Our framework provides a general mechanism for incorporating domain knowledge into learning. The framework offers the domain expert a guideline for making background knowledge specific and relevant: adapt it to suggest alternative representations. Though it remains the domain expert's responsibility to find ways of making the knowledge relevant to representation, there is often a fairly intuitive connection in many real-world domains. Our framework gives an algorithm for using knowledge translated into this form to improve learning. The algorithm is tolerant of uncertainty and can be used to improve an arbitrary induction algorithm.

- Finally, the framework facilitates interaction between a human expert and a learning program. The human can make hypotheses by generating a set of alternative representations and prioritizing them, which a system based on our framework could then search empirically to discover changes that improve induction. The results of searching the space of suggested representations might be summarized to critique the input knowledge, which a domain expert could use to refine his or her understanding of the domain.

Using an implementation of our framework on the protein structure prediction problem, we obtained the following results in biology:

- By re-representing protein sequences in terms of amino acid properties, the ability of homology modeling to predict protein structures (fold classes) can be improved.
- The current definitions for known properties of amino acids may need to be slightly revised for the purposes of homology modeling. For example, analysis of our results suggest that the set of small amino acids should be reduced (Dayhoff *et al.*, 1972), histidine belongs in the positive group rather than the aromatic group (Taylor, 1986), and glutamine may play a unique role (Richardson and Richardson, 1989).
- The relevant property at a site depends on its context, and this allows many more properties to be important besides hydrophobicity.
- The pattern of hydrophobicity among the three N-terminal and three C-terminal neighbors of each amino acid can be used an effective definition of context.
- When this notion of context is used to select the relevant property for transforming the amino acid at each site, the average z-score between proteins in the same fold class can be increased beyond that for homology modeling with amino acid identities (z-scores: identities=5.22, search=5.64), and even beyond the improvement made by using substitution tables in alignments (z-score: MDM78=5.60; Dayhoff *et al.*, 1978).

1.6 Overview

The rest of this thesis is divided into two main parts. The first half relates mainly to machine learning, reviewing related research (chapter 2), giving a formal description of our framework (chapter 3), and an describing an implementation called CORIL (chapter 4). Chapter 4 also contains an extended example that illustrates how domain knowledge may be captured in the formalism of the framework, and then how this knowledge might be translated for use by CORIL to actually solve the problem. The second half of this thesis consists of an application of the framework to the difficult real-world domain of protein structure prediction. We present the relevant background information in biology (chapter 5) and then describe several experiments in which an implementation of our framework was used to improve prediction in this domain (chapter 6). We conclude with an analysis of the framework at several levels (chapter 7).

Chapter 2

Related Machine Learning Research on Knowledge and Representation

A common characteristic of real-world domains that are difficult for learning is that the examples are given in a *low-level representation*. Intuitively, a representation is considered to be low-level when no single attribute has a strong relationship with the target concept. This property of low-level representations interferes with several pervasive biases that are common among nearly all known induction algorithms. As a consequence, many standard algorithms may learn poorly in such domains.

In the first part of this chapter, we review the role of representation in machine learning and discuss the negative effects of a low-level representation. Given, the problems that a low-level representation can cause, an approach frequently taken is to try to shift to a higher-level representation to improve learning. We will discuss some automated methods for changing representation, such as feature construction and constructive induction. We point out a ubiquitous problem that all automated methods for change-of-representation must face: dealing with complexity.

Another characteristic of real-world domains is that some background knowledge is often available. We would like to take as much advantage of this resource as possible to improve predictive accuracy. In the last part of this chapter, we review known roles for knowledge to play in machine learning, focusing on desirable criteria such as flexibility and generality.

2.1 The Role of Representation in Learning

In the context of machine learning, representation refers to the terms that are used to describe examples. For instance, in domains where examples are described by feature vectors, the representation is the sequence of features comprising the vector. Each feature can be treated as a range of values that expresses some information about each object. Different representations can be realized by adding or dropping features from the vector. Different representations can also encode the same information in different ways, such as by changing coordinate systems. Representation is an aspect of a whole space of example descriptions, so changes in representation must be uniformly applied to all descriptions of training and testing examples. Not all domains are based on feature-vectors, but their representations still use some sort of language to describe unique aspects of objects in the domain.

Inductive algorithms often use the terms in a representation to construct explicit internal hypotheses. For example, the candidate elimination algorithm forms intensional descriptions of classes (concepts) based on collections of terms (attribute-value pairs) that are common to all positive examples but do not cover any negative examples (Mitchell, 1982). Similarly, ID3 builds decision trees by recording tests on the value of a term that provide the greatest information gain when used to separate examples at each node (Quinlan, 1983). Our notion of representation is distinct from the internal hypothesis language, referring only to the language in which examples are presented to the learner. In fact, some induction algorithms do not even form explicit internal hypotheses, such as neural networks (Rumelhart *et al.*, 1986) and instance-based algorithms (Aha *et al.*, 1991).

The role that representation plays in machine learning can be explained in terms of bias (Utgoff, 1986). According to the classification model of learning, the goal of the learner is to find a partition of the domain objects, consistent with the training examples. In order for a hypothesis to be consistent, it is committed to classifying the training examples as specified, but the remaining instances in the space may be classified arbitrarily. To construct a complete partition, each class may be extended (generalized) from the specific set of examples of that class given during training to any superset, provided that the newly included instances are not known to belong to any other class. In general, there are many possible ways to generalize a set of training examples into a consistent partition, and bias is anything that influences which partition is selected for each set of examples.

Representation is a bias because it can affect how a set of examples is generalized. Certainly, representations that encode different information about the examples can cause different generalizations to be drawn. If some characteristic of an object that is strongly related to the target classification is dropped from a representation, then the algorithm may no longer be able to recognize the pattern in the examples. Hence it might choose an arbitrary partition that is consistent with the training examples but mis-classifies many other instances in the space, in comparison with the more accurate partition hypothesized when the feature is included. However, the performance of a learning algorithm is not necessarily a monotonic function of the amount of information encoded by the representation. For example, if too many superfluous features are included, an incorrect generalization might be drawn due to spurious correlations. Conversely, an example of improving accuracy by reducing information content is the discretization of numeric variables. Real-valued features are replaced with symbols for a finite set of ranges that are known a priori to be relevant to classification. Although discretization reduces the options for partitioning the examples based on the numeric attribute, it recommends a specific grouping of examples that can save the algorithm the effort of discovering this pattern on its own.

Varying information content is not the only way that different representations can produce different learning performance; even representations that encode the same information can cause an algorithm to generalize the same set of examples in different ways. For example, although computing a new feature from the original ones does not increase the information contained in the representation, it may extend the set of hypotheses that the algorithm can express. Whereas the algorithmic bias constrains the number of ways a set of examples can be generalized into a partition, often via a restricted internal hypothesis language, computed features can act as complex sub-expressions to create concept descriptions that could not be expressed with atomic symbols. For example, an algorithm that is designed to learn conjunctions of features can be adapted to learn a disjunctive concept by appending disjunctions of features to the representation. Similarly, computing the square or logarithm of a numeric variable can facilitate an algorithm like the perceptron by extending discrimination beyond linearly-separable partitions (Minsky and Papert, 1969). Saxena (1991) has presented a theoretical explanation of this interaction between representational and inductive biases based on data compression. Other theoretical work has looked at the effect of different representations on learning in terms of altering the concept-class complexity (Haussler, 1988).

2.2 Low-Level Representations

Because of interaction between the representation and the inductive algorithm, the utility of a representation for learning depends on the algorithm being used. So a given representation may produce adequate results with one algorithm but poor performance with another. However, recent empirical studies have suggested that representations in some real-world domains can be poor in a more general sense because a wide variety of algorithms fail to learn adequately with them. Rendell and Ragavan (1993) demonstrate several concepts that are all learned with similar inadequacy by a broad class of induction algorithms, including ID3, FRINGE, GREEDY3, BackProp, and MARS. Ragavan et al. (1993) present a more detailed study of the difficulty that many induction algorithms have in the domain of financial risk analysis. The representations in these domains can be blamed for the poor performance because a novel algorithm, LFC, achieved much higher accuracies–sometimes as much as 20 percentage points higher–by changing representations.

Many of these troublesome representations can be labeled as "low-level." There may be other causes of difficulty for learning in real-world domains, such as noise (Quinlan, 1986) or intrinsic inaccuracy (Rendell and Seshu, 1990), but we assume these are not a problem in the scope of this thesis. A low-level representation generally includes all the information that is necessary to make classifications, but individual parts of the representation are only remotely related to the target concept, making patterns hard to identify. A classic example of a low-level representation is the board-configuration of a chess game. Although concepts such as forced-win are completely determined by a specification of the pieces in each of the 64 board positions, the relationship is highly complex. The relationship between the target concept and the piece in any particular position is obscure and can change wildly with even small changes in the configuration of the rest of the chess board.

Low-level representations are a major cause of learning difficulty in real-world domains, especially where examples are naturally described by many small measurements. In this section,

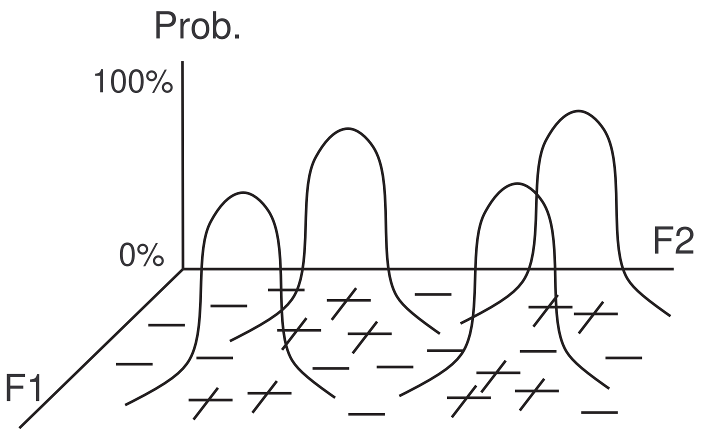

Figure 2.1: An illustration of concept dispersion. Separate clusters of positive instances in the plane appear as peaks. The horizontal axes (F1 and F2) are two dimensions used to describe the instances. The vertical axis gives the local probability that an instance in the area will be in the positive (+) class.

we explain why so many common algorithms have trouble learning with low-level representations. Concepts that are dispersed in the instance space inhibit many algorithms due to a common underlying bias. An analysis of dispersed concepts in feature-vector-based domains suggests that feature interaction is a primary cause of low-level representations in this common class of domains.

2.2.1 Concept Dispersion

One indicator of low-level representations is concept dispersion (Rendell and Seshu, 1990). Dispersion refers to the distribution of instances of a concept in the instance space. Although dispersion applies to concepts in any domain with a metric, one way of visualizing dispersion is to imagine the concept as a probability surface over a two-dimensional plane (see Figure 2.1). Peaks in the surface will rise above clusters of points that belong to the concept. Concept dispersion is a function of: 1) the number of peaks, 2) the shape of the peaks, and 3) the arrangement of the peaks. In general, concepts that have many odd-shaped peaks scattered about in instance space are difficult to learn.

The number of peaks, or peak multiplicity, is the primary factor in concept dispersion. A large number of peaks can make learning difficult for many algorithms (Rendell and Cho, 1990). The reason a wide variety of learning algorithms are affected in a similar way is that the all share a common underlying bias. Despite the differences among algorithms like ID3, GREEDY3, NT-Growth, and BackProp, they all implement the similarity-based learning (SBL) bias, tending to classify unseen instances like nearby instances that were seen during training (Matheus and Rendell, 1989). Nearest-neighbor learning algorithms implement the SBL bias directly (Aha *et al.*, 1991). This bias is very general and is pervasive among commonly known learning algorithms. For example, the SBL bias is implicit in the way a recursive partitioner splits the instance space into regions (Breiman *et al.*, 1984). Algorithms that implement the SBL bias perform best when the instances of a concept are clustered together into a few large peaks (E-space regularity; Rendell, 1985). If a concept is dispersed, instances of the concept will be found in many small clusters separated by instances not in the concept. This condition violates the assumption of the SBL bias because neighboring instances are less likely to have the same classification.

Another aspect of dispersion that can influence learning performance is the shape of the peaks of a concept in instance space. Many algorithms tend to learn concepts best if their peaks are generally convex. While some algorithms learn axis-parallel hyper-rectangles (e.g. ID3; Quinlan, 1983), others learn oblique hyper-planes (e.g. perceptron; Minsky and Papert, 1969), and still others learn multi-dimensional balls or ellipsoids (e.g. AutoClass; Cheeseman *et al.*, 1988), few algorithms can learn odd-shaped, concave objects easily. Figure 2.2 gives some examples of increasingly difficult shapes to learn in R2. Because of the SBL bias, such algorithms will tend to classify points in between two instances of the same class in the same way. Depending on the capabilities of the inductive algorithm, these concepts might be broken down into disjunctions of several smaller convex peaks, but this merely reduces to the peak multiplicity problem discussed above.

The final aspect of concept dispersion is the arrangement of peaks. One can think of a large peak in instance space as a collection of smaller peaks that are adjacent. SBL algorithms work best on concepts whose peaks can be merged together because there is a stronger correlation between the classifications of neighboring instances. But some arrangements of peaks, even though they are adjacent, cannot be easily merged. For example, the exclusive-or of two variables in a Boolean domain causes positive instances to be scattered diagonally throughout the space. Higher-order versions of this phenomenon, called parity, have been studied as a funda-

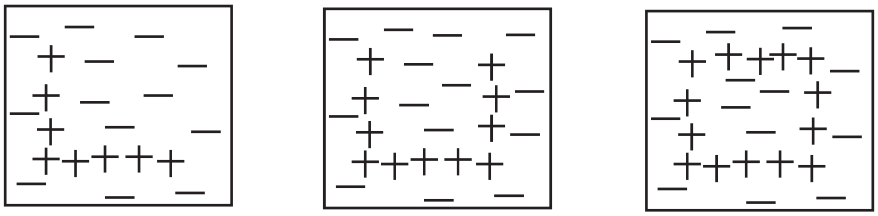

Figure 2.2: Examples of concave concepts that are increasingly difficult (left to right) for SBL algorithms.

mental source of difficult Boolean concepts (Seshu, 1989). In these situations, the instances of a concept may lie in a contiguous region, but the high-level pattern is hard for SBL algorithms to detect. Instead the concept must be learned as a collection of smaller peaks, again causing difficulty via peak multiplicity.

2.2.2 Feature Interaction

In domains represented by feature-vectors, concept dispersion is often a symptom of an underlying phenomenon called feature interaction (Rendell and Seshu, 1990). Feature interaction occurs when multiple parts of a representation must be considered together to observe the pattern in a concept. Specifically, the relationship between one feature and the target concept is affected by one or more other features. Thus feature interaction makes patterns particularly difficult to spot because they are conditional. The idea of feature interaction can be generalized to other kinds of domains not based on feature vectors, as long as the representation of examples consists of individual components that can be inter-related.

As an example of feature interaction, consider the prediction of heart disease from a database of patients' physiological attributes. The record for each person might contain measurements of systolic and diastolic blood pressures. It is well known that a person's cardiovascular health is not independently determined by the absolute values of either of these variables, which naturally vary due to blood volume and heart rate. Rather, it is the difference between these two values (pulse pressure) that is responsible for propelling the blood through the body. So the relationship between systolic pressure and heart disease depends on the diastolic pressure, and vice versa, implying that these two features interact.

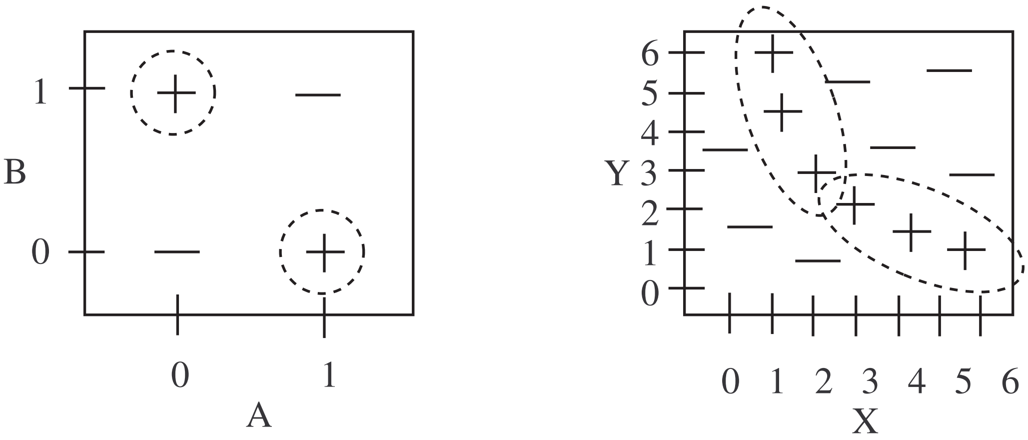

Figure 2.3: Two examples where feature interaction produces multiple peaks. The left picture is of the Boolean concept (A xor B), and the right picture is of the real-valued concept (X*Y=6).

Feature interaction can lead to concept dispersion by causing multiple peaks to arise in various parts of the space. Because of the conditional dependence of relationships among variables, a peak that shows the correlation between one variable and the target concept may "move" as the value of an interacting variable changes. In the space as a whole, clusters of positive instances appear to form separate peaks centered on different sets of values for the interacting variables. Figure 2.3 shows two prototypical examples of feature interaction: exclusive-or in Boolean domains, and an analog of it based on products of real-valued variables. Each of the variables in these examples is related to the target concept, but the nature of the relationship (i.e. location of the peak) depends on some other variable. For example, in the real-valued space there appears to be two distinct clusters of points. When X is low ($X < 2$), a broad range of Y values satisfy the concept, but when X is high, the pattern of vectors in the concept is restricted to those where $Y < 2$. Complex interactions among multiple features can produce a vast number of small, separated clusters of positive instances that each appear to be special cases based on elaborate conditions.

So feature interaction can cause learning difficulty through concept dispersion, especially via peak multiplicity and arrangement. Feature interaction is also problematic for learning algorithms in feature-vector-based domains because, in addition to sharing the SBL bias, these algorithms often make decisions by looking at only one feature at a time (greedy SBL; Matheus

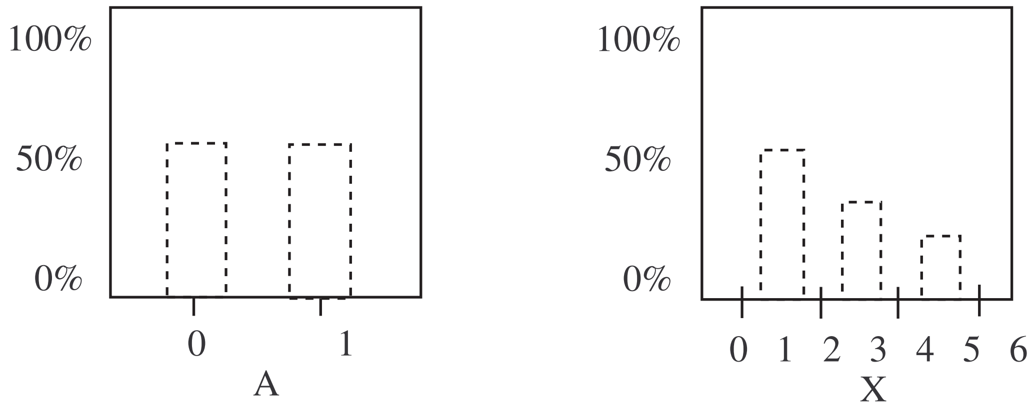

Figure 2.4: An illustration of blurring for the concepts in Figure 2.3. The instance spaces are projected onto the horizontal axes, showing the percentage of positive examples over all values of the interacting variable.

and Rendell, 1989. Many algorithms, from CART (Breiman *et al.*, 1984) to stepwise multiple regression (Draper and Smith, 1981), work by iteratively identifying the variable most correlated with a subset of the training sample. These local decisions are not guaranteed to be globally optimal; a more compact concept description that recognizes higher-level patterns might exist.

Abstractly, greedy SBL is equivalent to projecting all the instances from a multi-dimensional space onto each axis to look for patterns where similarity can be exploited. During projection, the relevance of an important variable might be masked because the peaks which are separated in the whole space by different values of an interacting variable get "blurred" together. Figure 2.4 shows the effect of projecting the examples in Figure 2.3 onto individual axes (A and X). Patterns of positive instances are much less clear than the well-defined peaks in two-dimensions. In fact, variable A in the Boolean example does not appear to be involved in the concept at all, since different values do not affect the proportion of positive instances. Rendell and Ragavan (1993) have used the notion of blurring to develop a data-based measure of concept difficulty that is based on the entropy of such projections.

Various attempts have been made to address the greediness of typical SBL algorithms, such as by using lookahead to search among more than one feature at a time for correlations with the target concept (Norton, 1989). But ultimately, the number of combinations of features is exponential, so it is intractable to capture arbitrary feature interactions. LFC uses beam search as a heuristic to explore this large space of potential features (Ragavan and Rendell, 1993). Still, feature interaction remains a hallmark of difficult, real-world domains for machine learning.

2.3 Change of Representation

Although representation can be blamed for poor learning performance in some domains, representation can also be part of the solution to improving performance. We have seen that any given set of examples can be represented in wide variety of ways, and different representations can affect generalization by interacting with the inductive bias. Choosing the right representation of a problem has long been known to have a significant impact on the performance of AI programs (Newell, 1969; Amarel, 1970; Korf, 1982). Several machine learning studies have also noted the importance of using the right representation for learning (Samuel, 1959; Winston, 1975; Flann and Dietterich, 1986). In this section, we discuss several learning systems that actively manipulate the representation of a domain to increase the accuracy of an induction algorithm (which is held constant).

2.3.1 Constructive Induction

In machine learning, automated methods for changing representation have generally been called it constructive induction (Michalski, 1983). Constructive induction typically refers to the extension of a representation by a new syntactic component that is a function of the original description. This new component is computed for all examples simultaneously, and used to augment their descriptions in some way. For example, the INDUCE program uses special transformation operators in annotated predicate calculus (APC) to generate new terms in its search for hypotheses to model a concept (Michalski, 1983). Similarly, Dietterich and Michalski (1986) describe a program that transforms sequences of moves in a card game by manipulating objects in the sequence to summarize key aspects of the surrounding moves.

A common constructive induction technique is to invent new terms, such as by clustering (Anderberg, 1973). Clustering can improve a representation by giving a unique label to separate peaks in instance space (Rendell, 1985). Examples of induction systems that exploit clustering are CLUSTER (Michalski and Stepp, 1983) and COBWEB (Fisher, 1987). Another method for inventing new terms is inductive-logic programming (ILP). ILP uses inverse resolution to augment a domain theory with a new predicate that covers some unexplained observations (Muggleton and Buntine, 1988).

2.3.2 Feature Construction

In the context of feature-vector based domains, much of the research on constructive induction has focused on a specialized class of techniques called *feature construction* (Matheus and Rendell, 1989). In feature construction, the initial representation of a domain is extended by computing a new feature as a function of the original feature values for each example, and then literally appending it as an extra component to each vector. New features can potentially improve learning performance in difficult domains by enhancing the metric in instance space. The new syntactic component might expose new similarities among the instances, effectively merging peaks (see Figure 2.5), and thus facilitating the broad class of SBL algorithms (Rendell and Seshu, 1990). Such helpful constructions have been called "high-level features" because they are more directly related to the target concept, making patterns easier to detect. Another perspective is that constructed features can improve learning by acting as intermediate concepts, such as center control and knight fork in chess (Quinlan, 1983).

The type of feature construction operators available depends on the data types of the combined features. For Boolean features, logical operators like AND and OR can be used. For real-valued features, mathematical operators like MINUS and TIMES are commonly used. However, binary operators allow a quadratic number of possible constructions. Furthermore, if constructed features may be used in other constructions, then the number of possibilities is exponential in the number of original attributes. So the challenging aspect of feature construction is deciding which features to combine.

FRINGE exploited the replication problem in decision trees to select features of construction (Pagallo and Haussler, 1990). This involves looking for patterns of decisions that are frequently repeated at the leaves of a decision tree, which indicates that the composite variable might make a good choice higher up in the tree. STAGGER estimates the utility of a potential construction based on the proportion of positive instances the new feature would cover versus the proportion of negative instances it would not cover (Schlimmer, 1987). LFC uses a similar function of instances covered, based on a geometric argument of which new features could have the lowest

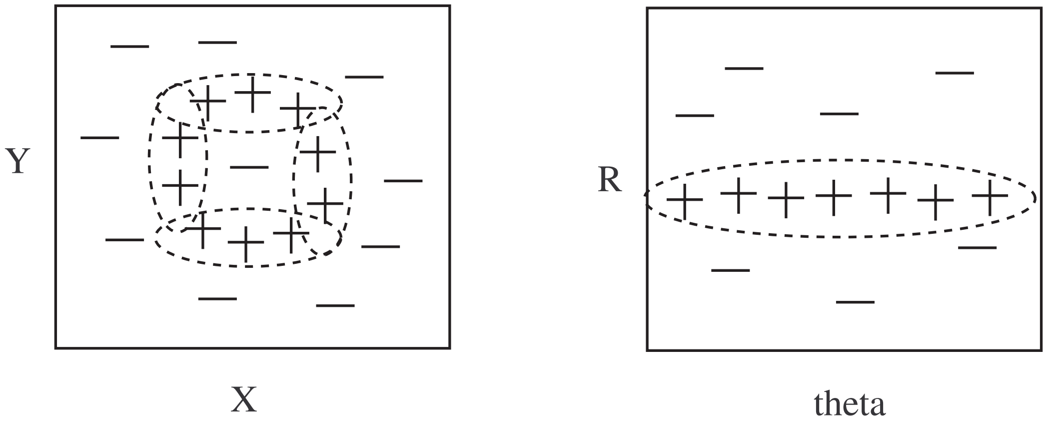

Figure 2.5: An example of how changing representations can merge peaks in instance space, and thus improve the ability of an SBL algorithm to learn the concept accurately. In this case, the instances plotted in their original representation (X, Y) are scattered in instance space, creating several small, dispersed peaks. The representation is changing to polar coordinates: the original variables X and Y are replaced with the computed variables $R = sqrt(X^2 + Y^2)$ and $theta = tan^{-1}(Y/X)$. This causes the instances to shift in the space so that there are fewer, larger peaks.

entropy (Ragavan and Rendell, 1993). BACON decides which features to construct by looking for products and ratios of numeric variables that are nearly constant (Langley *et al.*, 1983).

2.3.3 Dynamic Bias

Another way of looking at change-of-representation in machine learning is as a form of *dynamic bias* (Utgoff, 1986). In this view, representation is seen as an aspect of bias, in the sense that it maybe changed independent of the examples themselves to affect generalization. One of the earliest programs to use dynamic bias was STABB (Utgoff, 1986), which used constraint-propagation through problem-solving transcripts to suggest new attributes for describing problems in order to prevent a version-space from collapsing. Tcheng et al. (1989) propose a more general system for optimizing the bias of a learner, which could include representations as well as arbitrary parameters of the algorithm. Finally, Gordon and DesJardins (1995) present an abstract, multi-layered model of dynamic bias in which an induction algorithm is incrementally improved by searching through a space of potential biases, which again subsumes change-of-representation. However, this model refers to arbitrary biases, and its application to representation is somewhat trivial.

2.3.4 Related Methods in Statistics and Pattern Recognition

Change-of-representation is not unique to machine learning and has arisen independently in other fields, such as statistics and pattern recognition. Devijver and Kittler (1982) propose several heuristic techniques for *feature selection*, which reduces dimensionality by finding a subset of variables that separates classes well. This approach is similar to the use of stepwise multiple regression for selecting variables to include in a linear model (Draper and Smith, 1981). Factor analysis can also be used to find a minimal set of variables that explains a maximal amount of variance in a dataset (Harman, 1976).

Devijver and Kittler (1982) also propose methods for *feature extraction*, in which multiple features are summarized; some of the information lost in deleted features is incorporated in new (computed) variables. Feature extraction can transform the space in a fundamental way (more radical than simple projection, for example axis rotation) that might be necessary for detecting certain patterns (Duda and Hart, 1973). New variables might be linear combinations of original features, as in principal component analysis. Or the examples might be completely re-described over a new set of basis features, such as in a Fourier transform (Mansour, 1994). Multidimensional scaling (MDS), which derives a new set of basis features from analysis of the differences among examples, can also be seen as a method of changing representations (Davison, 1983).

2.3.5 The Complexity of Change-of-Representation

A ubiquitous problem that all methods for change-of-representation must find a way to deal with is *complexity*. In general, there are usually an infinite number of ways to represent a given data set. If one restricts attention to simple feature construction, then even a binary operator (such as AND or PLUS) can generate a quadratic number of new features. If newly constructed features are allowed to be candidates in subsequent constructions, then there can be an exponential number of extensions to an initial feature vector (Matheus, 1989).

Each of the automated methods for change-of-representation we discussed above has some specific way of dealing with this complexity. Typically, constraints on which changes-of-representation will be considered are implicitly built into the algorithm. For example, FRINGE uses the replication problem in decision trees to help decide which features should be combined (Pagallo and Haussler, 1990). And stepwise multiple regression uses a greedy method for testing and selecting variables to add or drop one at a time, based on their effect on the variance of the model (Draper and Smith, 1981). These techniques do not guarantee finding the best representation, and are unable to cover the whole space of possibilities, though they have been found to be effective heuristics in improving predictive accuracy.

We would to see these difficult decisions made more intelligently The constraints on change-of-representation used by all these methods are different, and they each rely on unique assumptions about which kinds of change-of-representation will be useful for a particular domain. We would like to see these assumptions made explicit, for example based on domain knowledge. This would unify a lot of these methods, showing how they are similar in a deeper sense. Revealing the underlying similarities among these disparate methods could also benefit other domains, where change-of-representation might be useful, but existing techniques are not applicable. For example, Hirsch and Japkowicz (1994) give a domain-specific method for improving the prediction of promoters by incrementally augmenting the representation of DNA sequences with special "summary" features. A more general framework for change-of-representation is needed that captures the role that domain knowledge can play in reducing complexity by helping decide which changes in a large space of possibilities to consider.

2.4 Related Research on Knowledge in Machine Learning

Another common characteristic of real-world domains is domain knowledge. Learning almost always occurs within some context (e.g. scientific), which can provide some additional background information. A wide variety of things may be known that are in some way relevant to the concept being learned. One might know spatial relationships among the attributes measured, such as proximity of positions on a chess board (Ragavan and Rendell, 1991). One might have a plausible model for a causal mechanism that underlies the phenomenon (Rajamoney, 1990). Intermediate concepts or knowledge of related concepts might be available (Fu and Buchanan, 1985). Knowledge about the data itself, such sample bias, precision of the instruments, or critical cases might be known (Gaines, 1989). Another class of knowledge relates to properties of the target concept, such as degree of disjunction (Valiant, 1984).

Most induction algorithms are not designed to make use of such domain knowledge. Standard learning algorithms often attempt to construct generalizations from a set of examples alone. Yet in real-world domains, we want to take advantage of every resource available to construct predictive models that are as accurate as possible. Pragmatically, knowledge is sometimes necessary to make up for small databases, which might be fixed due to the high cost of collecting examples in a real-world domain. Thus it is important to develop techniques for incorporating knowledge into learning. In this chapter, we look at several roles that are known for knowledge to play in learning, and discuss desirable properties for such methods for using knowledge.

2.4.1 Implicit Ways Knowledge is Already Used in Machine Learning

Knowledge is implicitly used in almost all learning experiments in several ways. First, domain knowledge is often used to choose the induction algorithm. Different induction algorithms have different biases, each of which guides a learning algorithm in a unique fashion away from some hypotheses and toward others. Thus a good bias for one domain can actually interfere with learning in another domain. Each algorithm has certain criteria for applicability, such non-disjunctiveness for version spaces (Mitchell, 1982), or linear-separability for perceptrons (Minsky and Papert, 1969), or smoothness of class functions for instance-based learners (Aha *et al.*, 1991), or independence for Bayesian learners (Cheeseman *et al.*, 1988). The appropriateness of the bias of a given algorithm for a particular domain depends on that domain; so the domain expert is ultimately responsible for making a reasonable choice.

In a similar manner, domain knowledge is often used to select the parameters for the induction algorithm. Many algorithms have parameters that must be specified, such as the stopping criterion of ID3 (Quinlan, 1983), the learning rate of BackProp (Rumelhart *et al.*, 1986), or the number of neighbors maintained for each cluster in an instance-based learner (Aha *et al.*, 1991). The accuracy of a learner can be sensitive to the choices for such parameters, so domain experts often use background knowledge to help make reasonable selections. However, choosing an algorithm with an appropriate bias and selecting reasonable parameters for that algorithm are fairly awkward ways of incorporating domain knowledge, often requiring a deep understanding of the induction algorithm itself. Hence a common approach is to use an automated procedure to search the space of parameters of a learning algorithm for optimal settings, which neither requires a comprehension of how the algorithm works, nor uses any specific knowledge about the domain (Tcheng *et al.*, 1989).

Finally, knowledge is naturally used to determine the initial representation of examples in a domain. A great deal of knowledge can be utilized in initially choosing how to describe examples. One of the primary decisions concerns what attributes to measure, so that all the information relevant to the target concept will be encoded in the data. Other options include choice of coordinate system, relative versus absolute values, logarithmic versus linear scales, how many significant digits to record for numeric attributes, and splitting multi-valued nominal attributes into several Boolean features. A common technique, especially with symbolic learning algorithms, is to discretize numeric features, which requires specific knowledge of reasonable cutoffs for ranges that are likely to be relevant. Unfortunately, representations often need to be compact to conserve storage space, and, for example, any attributes that can be computed, though they could be potentially useful for learning, might be left out of the initial representation.

2.4.2 Explicit Roles for Knowledge in Learning

Selecting an algorithm with an appropriate bias, or reasonable parameters for the algorithm, is an awkward way of expressing knowledge in machine learning. Several explicit roles for incorporating knowledge more declaratively in learning are known. Some roles for knowledge consist of enhancements to standard induction algorithms, such as KBANN (Towell *et al.*, 1990) or PROTOS (Porter *et al.*, 1990). Others roles introduce novel learning algorithms to accommodate the knowledge, such as explanation-based learning (DeJong and Mooney, 1986) or analogy (Rajamoney, 1990). Still others roles for knowledge are more general, and can be used with a wide range of algorithms, such as Abu-Mostafa's (1994) method of providing "hints." In the most general sense, knowledge plays a role as a bias, and we discuss some research on this perspective.

The most direct way to incorporate knowledge into learning is to enhance known induction algorithms so that they can make use of the knowledge. For example, PROTOS improves the ability of a case-based learner to index cases by letting the knowledge suggest attributes to use as remindings for cross-linking the cases (Porter *et al.*, 1990). KBANN can improve the performance of training a neural network by pre-setting the weights on connections so that it has a topology that reflects the structure of the domain theory (Towell *et al.*, 1990). Another example of a method for using knowledge to improve a known learning technique is MIRO, which constrains the features generated by a feature-construction algorithm by filtering out all new features except those that can be deductively derived from the domain theory (Drastal *et al.*, 1989).

Explanation (EBL) is another role that knowledge can play in learning (DeJong and Mooney, 1986). This approach is useful when the domain knowledge can be used to explain why examples are classified the way they are. For example, if the domain theory is in the form of sentential logic (e.g. Horn clauses), then a proof-tree for each instance is constructed. The importance of the explanation structure is that it reveals the aspects of the example on which the observed classification depends. The description of a single instance may be transformed into a general concept description by dropping irrelevant details.

If knowledge is available about similar concepts, possibly even in other domains, then this knowledge might be exploited via analogies. Several methods have been proposed for learning from analogies. One well-known approach is structure-mapping, which refers to the assumption that, aside from the specific names for terms in the new domain, the syntactic relationships among those terms is parallel to that in the old domain (Falkenhainer *et al.*, 1989). Thus detailed knowledge of how one domain works, plus a belief that is it similar to a new domain, plus an plausible mapping between terms in the two domains, can produce hypotheses in the new domain that follow the same pattern as in the old. Davies and Russell (1987) propose another method for constructing analogies within a single domain, called determinations. Determinations are beliefs about what attributes (or relations) determine the values of what other attributes. Knowledge about the details of one case may be used to make inferences about another case provided the similarities between them support the determination relationships.

Abu-Mostafa (1994) proposes a general method for using knowledge to generate artificial examples as "hints" to illustrate some property of the target concept. For example if the concept is known to obey some invariance with respect to a particular variable, then additional negative examples may be generated that are identical to an original example except for having different values for this variable. The point of this approach is to simulate a teacher who designs a particular sequence of examples to lead the learner to realize some concept.

Theory revision may also be looked at as a way of incorporating domain knowledge into learning (Pazzani, 1988). There are several perspectives on theory revision, such as the incremental refinement of a theory that fails to explain a few examples (Bergadano and Giordana, 1990), or as an integration of EBL and SBL (Mooney and Ourston, 1994). However, given that the inputs to theory revision include both examples and knowledge, it can also be thought of as method for incorporating knowledge into inductive learning. In this view, the domain theory provides a very strong bias on the hypotheses that can be generated. Using techniques like abduction, theory revision systems generally try to make minimal changes to the original theory to cover the most unexplained examples. This constraint allows nearly correct generalizations to be derived from a small number of examples. Inductive-logic programming (ILP) also fits in this category, since new terms are invented to cover examples with the help of a domain theory by the process of inverse resolution (Muggleton and Buntine, 1988).

More generally, the role knowledge plays in learning can be explained in terms of bias (Mitchell, 1980). Bias refers to anything other than the data itself that is used to select hypotheses during generalization. Biases are often implicitly built into induction algorithms. However, Dietterich's (1986) knowledge-level analysis suggests that the bias in an algorithm is equivalent to a prior state of knowledge which gets augmented during learning. While Dietterich conjectures that the knowledge in many biases cannot be captured deductively (i.e. in a first-order logic theory), Russell and Grosof (1987) show how many biases can be interpreted explicitly as non-monotonic beliefs. Turning this argument around, knowledge that improves learning in some way can be thought of as bias because it influences which hypotheses are selected among those that are consistent with the observed data. Michalski (1983) shows how this proposal may be implemented by encoding statements about how to construct plausible concept descriptions in annotated predicate calculus.

2.4.3 Desirable Criteria for New Roles for Knowledge in Learning

Knowledge in real-world domains can be quite heterogeneous, and may not fit easily into one of the known roles, such as those listed above. Thus we are always looking for new ways of incorporating domain knowledge into learning. Any new roles for knowledge should be *general*, in the sense that they can work with a wide variety of induction algorithms. Some of the known roles for knowledge only work with specific algorithms. For example, KBANN only works with neural nets (Mooney and Ourston, 1994). However, we want to allow the domain expert to use whatever algorithm he or she believes has an appropriate bias for the domain. This could be especially important in real-world domains where a domain-specific method is already available that out-performs all other prediction schemes. An example of the desired level of generality is the technique of generating artificial examples to illustrate invariance or symmetry properties, which can be used with any induction algorithm (Abu-Mostafa, 1994).

Another criterion that would be desirable for any new role for knowledge is that it is *flexible*, particularly in dealing with uncertainty. Knowledge, especially in real-world domains, can be uncertain, such as being incorrect, or incomplete, or inconsistent (Wilkins, 1990). It is extremely difficult to state correct knowledge about the real-world, including all possible consequences, with qualifications and for every conceivable situation (i.e. the frame problem; Ford and Hayes, 1991). Some of the existing roles for knowledge are known to be sensitive to uncertainty. For example, wrong predictions may be made by a Bayesian method if it is given inaccurate estimates of prior probabilities (Cheeseman *et al.*, 1988). As another example, the original EBL method was criticized for being unable to learn with an incomplete domain theory (DeJong and Mooney, 1986). We want any new roles for knowledge to be more *tolerant of uncertainty*, such as approaches based on plausible inference (DeJong, 1994).

Chapter 3

A Search-Based Framework for Change-of-Representation in Learning

3.1 Overview of the Framework

In this chapter, we present a search-based framework for change-of-representation in machine learning. The representation of examples is seen as an independent dimension of learning that can be manipulated separately from the inductive algorithm. We assume a particular induction algorithm has already been selected and will be held constant throughout the learning process. The goal of our framework is to find representations that improve the accuracy of the learner, in comparison to the initial representation of examples.

While a variety of learning methods can be considered as doing change-of-representation, and the general idea of treating change-of-representation as search seems intuitive, our framework works out many of the formal details that have not yet been specified. The framework has three main components: the search space, an evaluation mechanism, and a strategy layer (see Figure 3.1). To describe the search space formally, we must define what we mean by "representations," which will be nodes in the search space. We intend for our framework to be applicable to a wide range of domains, so our definition of representation is more abstract than just "a list of features." To compare various nodes in the search space and monitor our progress toward the goal, we must have a mechanism for evaluating representations. This function is carried out by the evaluation component, which empirically estimates the effect of a change-of-representation on the accuracy of the learning algorithm by training and testing it on some examples transformed into the new representation.

Finally, every search procedure needs a method for determining the order in which nodes in the space will be explored. This function is performed by invoking the third component in our framework—the strategy layer. It is this component that can be used to incorporate domain knowledge, in particular to help decide which alternative representations might improve learning performance. We will present a knowledge representation formalism based on the semantics of suggesting alternative representations.

Pragmatically, a system based on this framework would be used in the following way. First, a domain expert would obtain some data (examples and counter-examples of the target concept) and select an induction algorithm with a bias that is thought to be appropriate. Presumably, running the algorithm on the data in its original representation would result in low accuracy (a high error rate). Then the domain expert would come up with some domain knowledge that is believed to be relevant to learning the target concept, and state it in terms of representation. The knowledge would be encoded in the formalism we present and given as a domain theory to the system. Depending on how representations are described, the expert must also give a transformation function that can be used to syntactically convert an example from its original representation into a new representation.

When the system is given these resources from the environment—a domain theory, a transformation function, an induction algorithm, and a set of examples—it may begin its search. The start node for the search is defined by the initial representation. The strategy component is then queried to determine the next most reasonable representation to consider, given the current state of the search. This representation is passed to the evaluation component, which first transforms the examples into this new representation. Then the data are divided into training and testing sets and given to the algorithm to estimate the effect on accuracy, such as by cross-validation. This score is used to update the search, and the process iterates, perhaps stopping when a representation that significantly increases the accuracy of the learner is discovered.

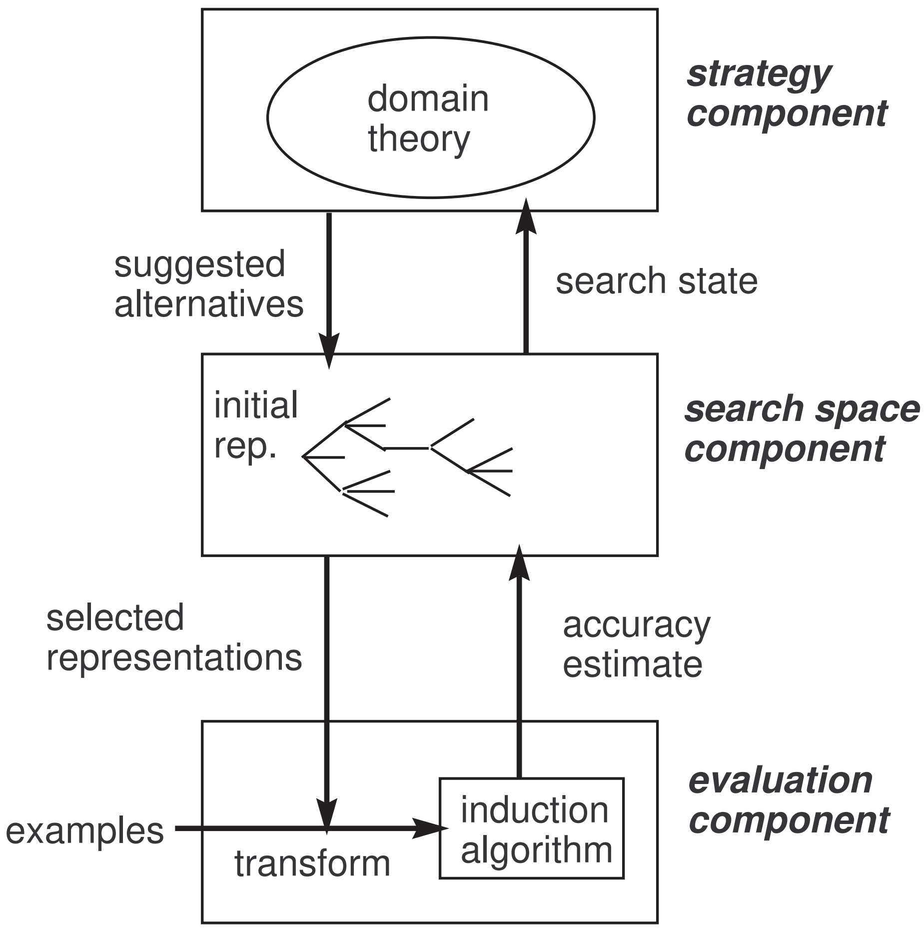

Figure 3.1: Search-based framework for change-of-representation. Change-of-representation is seen as an attempt to improve the accuracy of fixed learning algorithm by searching for alternative representations of the data. The central component is a search space consisting of alternative representations according to our formal definition. The evaluation component compares nodes in the space by transforming a set of examples into each new representation, training and testing the induction algorithm, and returning an estimate of the effect on accuracy. Finally, a strategy component that incorporates domain knowledge is used to determine which alternatives are worth considering at each stage during the search.

The whole motivation behind treating change-of-representation as a search process is that it provides a systematic role for knowledge to play in learning. Knowledge can be seen to act as a heuristic in the strategy component by guiding the search process toward alternative representations that are more likely to improve learning. We discuss the generality and flexibility of this role for knowledge, and give some examples of how several specific kinds of knowledge may be incorporated via our formalism. We also discuss how the framework might be implemented, and propose a system called CORIL that can be applied to a subset of domains. Finally, we comment on the relationship between our framework and various other machine learning systems.

We conclude this chapter by proposing how our framework might be extended to a more interactive learning system. We show how it is possible to use a system based on our framework to evaluate knowledge based on the results of the search. This can lead to a new class of results by providing the expert with high-level feedback, which might be use to refine their understanding of the domain. This ability can lead to interactive learning, in which the expert makes some hypotheses about the domain, tests them by running the search procedure, examines the results, and iterates until a good domain theory is converged upon. Such interaction is an important consequence of our framework, and might be necessary for effective learning in difficult real-world domains.

3.2 The Search Space Component

To describe the search space, we must give a formal definition for "representations," which constitute nodes in the space. Our definition is intended to be more abstract than typical definitions of representation in machine learning, which assume that examples are simply given as a "list of features." Such a definition applies only to domains where examples are initially given as vectors of a fixed number of attributes. However, many real-world domains are not easily forced into such a constant format. We want our framework to be applicable to a more diverse set of domains, where examples might initially be described in such variable-length forms as trees, graphs, sentences, etc. For example, in our application domain, protein structure prediction, examples consist of amino acid sequences, which vary in length from 100 to 1000 amino acids long.

We define a representation to be a pair $\langle L, D\rangle$, where L is a description language, and D is a denotation function. By language, we mean a set of symbolic objects, typically generated by a grammar. The grammar determines what elements exist as members of the language, generally via special combinations of primitive symbols. In addition, the grammar gives the language elements an internal structure, which can be used for defining syntactic relationships among descriptions. Because it is possible to generate some unwieldly languages using an arbitrary grammar, we make a further stipulation that there must exist an injective mapping F into finite strings of 0's and 1's ($F : L \mapsto \{01\}^*$), in order to maintain a concrete notion of "representability."

The denotation function is a mapping from domain objects $\mathcal{O}$ into the language:

$$D : \mathcal{O} \mapsto L$$

$$o_1 \neq o_2 \Rightarrow D(o_1) \neq D(o_2) \text{ for } o_1, o_2 \in \mathcal{O}.$$

Hence we say that the denotation function is responsible for "describing" objects in the real-world. We require the denotation function to be one-to-one, so that a unique description is given to each object in the domain. However, the denotation function does not necessarily have to be onto, because there may be some elements of the language that simply do not describe any possible object ("illegal descriptions"). This definition of representation differs from Korf's (1980) syntactic notion based on a data-declaration language. It is more similar to Holte and Zimmer's (1989) definition based on category theory.

As an example of a representation in this formalism, consider the longitude and latitude coordinates for cites on the globe. In this case, the language L consists of all pairs of real numbers: $\Re \times \Re$. The denotation function D is the mapping of each city into a specific pair of real numbers, based on their physical location. Other representations can map the locations of cities into different descriptions, such in spherical coordinates (azimuthal angles) or distances from three fixed points.

Based on this last observation, it is now convenient to define a *transformation function*, which can be used to relate various representations. A transformation is a mapping of the elements of one language into the elements of another language:

$$T : L_1 \mapsto L_2$$

$$T(a_1) = a_2 \in L_2 \text{ for } a_1 \in L_1 \in L_1.$$

By applying a transformation to all the elements in one language, a second language may be derived:

$$L_2 = T(L_1) = \{T(l) : l \in L_1\}.$$

Furthermore, by composing the original denotation function with the transformation function, a new denotation function is automatically recovered:

$$D_2 = T \circ D_1$$

$$D_2 : \mathcal{O} \mapsto L_2.$$

The new language and new denotation function together make a new representation in the formal sense:

$$R_2 = \langle L_2, D_2 \rangle = \langle T(L_1), T \circ D_1 \rangle.$$

Transformations are useful for generating new representations from the original representation of examples by purely syntactic manipulations. For example, a transformation from Cartesian coordinates $R_1 = \langle X, Y \rangle$ into polar coordinates $R_2 = \langle r, \theta \rangle$ can be achieved by the transformation function

$$T(\langle X, Y \rangle) = \langle sqrt(X^2 + Y^2), tan^{-1}(Y/X) \rangle.$$

3.3 The Evaluation Component

The goal of the search is to identify an alternative representation that improves the performance of the underlying induction algorithm. Our framework uses an evaluation component to monitor the progress of the search toward this goal. Our approach, similar to Schaeffer (1993), is to empirically measure the effect of various alternative representations by running the induction algorithm on re-represented examples and estimating performance via cross-validation. The most common performance measure for a learning algorithm is its *predictive accuracy*, which refers to the rate at which the learner makes correct predictions. In the environment, we assume there is a space of possible examples $\mathcal{X}$, and a stochastic process $\mathcal{P}$ that generates examples in some unknown sequence,

$$\mathcal{P}(1) = x_1, \mathcal{P}(2) = x_2, ...\mathcal{P}(i) = x_i, ...$$

where $i \in \mathbf{N}$ and $x_i \in \mathcal{X}$. Further, we assume that process is stable, such there exists a distribution $\mathcal{D}$ such that any subsequence (a sample of the process, indexed by a subset of the natural numbers) will converge on $\mathcal{D}$ as its size approaches infinity:

$$\lim_{|I| \to \infty} \{\mathcal{P}(i) : i \in I \subset \mathbf{N}\} = \mathcal{D}.$$

The accuracy is taken to be the frequency of correct predictions expected over arbitrary samples:

$$A = \langle (\frac{1}{|I|}) \sum_{i \in I} 1_{\{Pred(x)=Class(x)\}}(\mathcal{P}(i)) \rangle_{I \subset \mathbf{N}}$$

For conciseness, we use the notation for characteristic functions $1_{\{C(x)\}}$, which evaluate to 1 for all x that satisfy the constraint C(x), and 0 otherwise. 'Pred(x)' means the value predicted for example x, and 'Class(x)' means its actual value. Averaging over samples from the distribution for the process differs from the measure proposed by Devijver and Kittler (1982), who integrate over subsets of the instance space $\mathcal{X}$, which does not take into account the influence of the environment in which the learner is functioning.

For learning algorithms, accuracy depends fundamentally on the training set. In real-world domains, we are often given a fixed database of examples, which was perhaps costly to build, and asked to make the most out of this resource. We want to construct the best system based on this data for classifying other instances arbitrarily selected from the distribution. Thus accuracy will thus be a function of the training set S, and this can be incorporated into the formula via a conditional:

$$A(S) = \langle (\frac{1}{|I|}) \sum_{i \in I} 1_{\{Pred(x)=Class(x)|S\}}(\mathcal{P}(i)) \rangle_{I \subset \mathbf{N}}.$$

This approach does not capture the general performance of a learner, as is commonly done with learning curves that average expected accuracies over multiple samples as a function of training set size (Kadie, 1995). Rather, our pragmatic focus is on assessing the ability of various learners to make the best generalizations from a given set of data.

We cannot know the "true" error rate of a learner, since by assumption we do not have unlimited access to the distribution sampling test cases. However, we can *estimate* the performance of the learner by testing it on a random subset of the available data, which should be a sub-sample of the distribution. We will train and test the algorithm on specific sets of data, S_1 and S_2:

$$A^*(S_1, S_2) = (\frac{1}{|S_2|}) \sum_{x_i \in S_2} 1_{\{Pred(x)=Class(x)|S_1\}}(x_i).$$

It is important to use disjoint subsets to avoid over-fitting (2/3 for training and 1/3 for testing is commonly used; Weiss and Kulikowski, 1991). Over-fitting occurs when the learner tunes itself to the specific examples in the training set, without actually learning any generalizations. This can give artificially high accuracy estimates if the same examples are used for testing as for training (i.e. lower "re-substitution error;" Devijver and Kittler, 1982). If the distribution happens to generate repetitions of some instance, then syntactically identical examples might wind up in both the training set and testing set. However, the examples will be treated distinctly based on their unique positions (indices) in the stochastic process. This overlap is fair because it is representative of the performance environment.

There are two problems with the evaluation mechanism as developed so far. First, the accuracy estimate might be sensitive to the particular way the data happened to be divided, and accuracy is supposed to be independent of the test set. Second, withholding a subset of data for testing means less data is available for training, which could under-estimate the learner's potential accuracy when given the entire dataset. To solve these two problems, we could use *cross-validation*, which involves multiple iterations of this process of randomly dividing the data set and training and testing the learner (Stone, 1974). The data is first partitioned into k balanced, disjoint subsets or "folds."[1] The learner is trained on $(k-1)/k$ of the data, withholding one of the folds, and then tested on the $1/k$ unseen examples. This is process is repeated for each of the k folds to compute an average accuracy,

$$A^{CV}(S,k) = (1/k)\sum_{i=i}^{k} A^*(S \backslash S_i, S_i),$$

where the S_i are the random subsets of S. The more folds used, the closer the estimate gets to the "true" accuracy, though there is a corresponding increase in run-time complexity since each fold requires an entire run of the underlying induction algorithm. As an additional benefit, the set of accuracy scores generated by cross-validation can be used to compute a confidence interval on the average accuracy, which can help determine the statistical significance of any observed improvements (over the accuracy achieved with the original representation).

This approach generalizes to other performance measures besides accuracy. For example, these formulas could be used for error rate almost unchanged, except for replacing $Pred(x) = Class(x)$ with $Pred(x) \neq Class(x)$. The search component would simply have to be told

[1] Typical choices are $k = 10$ ("10-fold cross-validation") or $k = n$ ("leave-one-out cross-validation").

whether to minimize or maximize the resulting scores. Real-world constraints may also impose a domain-specific performance criterion. For example, if there is a differential payoff for getting certain classifications correct, then a weighting scheme might be used to bias accuracy in a way that is relevant. But the general principle of training and testing the learner on different subsets of data still holds, regardless of how performance is measured.

As with any learning algorithm, an important concern is whether the learner is over-fitting the data (Weiss and Kulikowski, 1991). In our case, the search for improved representations might over-fit the data by tuning the representation to aspects of the particular examples in the dataset. The use of cross-validation within each evaluation cycle does not ameliorate this problem because the sequential choices of representation between evaluations depend on cross-validation scores over the same set of examples, which violates the assumption of independence among samples. Thus, we are obligated to perform one more global level of cross-validation over the entire search process to fairly assess the significance of any apparent improvement in accuracy achieved by the best representation found.

3.4 The Strategy Component

In this section, we discuss the strategy component of our framework. As with every search process, we our change-of-representation framework needs a way of deciding the order of nodes to be explored during the search. There are many general strategies for searching, but these are "weak methods" since they do not take advantage of any special properties of a given domain (Pearl, 1984). We propose using domain knowledge explicitly to decide what node should be expanded next at each stage in the search. This reduces the complexity of change-of-representation by steering the search in a potentially huge space of alternatives toward regions that are more likely to contain improved representations. Thus knowledge is plays the classic role of a *search heuristic* (Newell and Simon, 1972). We first give some examples of how knowledge can guide the search for alternative representations, and then we present a semantics that formalizes this role for knowledge.

Domain knowledge can act in several ways to guide the search. First, knowledge can limit the space of alternatives by specifying a class of syntactic changes that are worth considering. This is most clearly illustrated in feature construction in the phase of operator selection

(Matheus, 1989). By choosing a particular operator, such as PLUS or XOR, the space of possible alternative representations is restricted to those which include only those new features that can be constructed from the original attributes using this operator. This can be a significant use of knowledge in real-world domains, since there are many possible feature construction operators. For example, specific knowledge about a domain might suggest that ratios of variables are the appropriate thing to look at, rather than differences, averages, products, exponentiations, etc. Such knowledge can be thought of as restricting the boundaries of the search space.

Another kind of knowledge might place more specific constraints on the form of representations that are expected to improve learning. For example, knowledge about data types can suggest that it might be inappropriate to subtract a variable in MILES units from a variable in HOURS units. If the OR operator is being used in a Boolean domain, then there might be a belief about a reasonable number of occurrences of this operator, corresponding to the expected degree of disjunction in the target concept. This role basically acts as a filter to steer the search away from bad representations.

Knowledge can also be thought of as serving a more positive role in the form of preferences that can be used to generate good alternatives. If spatial relationships among the attributes are known, such as on a chess board, then these might be the right ones to combine together in a feature construction. Or if intermediate concepts are known that might be relevant, it might be suggested to try appending these directly to the initial representation. Discretization can also benefit from domain knowledge, such as beliefs about relevant ranges for variables, as well as the likely number of such ranges in which to discretize a variable.

All of these examples are instantiations of a more general role for knowledge exposed by our framework, which is to it suggest alternative representations. That is, given the current state of the search, knowledge acts as a set of beliefs about the utility of other representations that have not yet been evaluated. To formalize this notion, we must first define the state of the search, for which we will use *partial functions.* Let $\mathcal{R}$ stand for the space of alternative representations. Let S be a function that maps representations into accuracy scores, that is the real numbers. At the start of the search, however, the performance of none of the representations is known So we add one extra element, $\perp$, to the range of this function to represent unknown accuracy scores. Thus, the functional specification is:

$$S : \mathcal{R} \mapsto [0, 1] \cup \{\perp\}.$$

Initially, $S(r) = \perp$ for all $r \in \mathcal{R}$. But with each step i of the search, a new node is explored, which updates the state S_i by mapping another representation to a specific accuracy (real number in the unit interval).

In our framework, knowledge is applied to suggest plausible accuracies for representations whose accuracies are unknown and are hence mapped to $\perp$ in the current state. For example, given that the first 16 representations have accuracies ranging from 65% to 70%, a expert might have reason to believe that the accuracy of the learner when examples are transformed into the representation r_{17} will be 72%. This kind of knowledge can be modeled in terms of *extensions* of the partial functions encoding the search state. An extension S_j of a state S_i (for which we use the symbol '$\succ$') maps all known representations to their original accuracies, but may also map some new representations to real numbers instead of $\perp$:

$$S_j \succ S_i \stackrel{def}{=} \forall_{r \in \mathcal{R}} (S_i(r) \neq \perp) \rightarrow (S_j(r) = S_i(r)).$$

When a domain expert expresses some knowledge, this can be thought of as a constraint on possible extensions for the true state of the world S_{true} (where the accuracies for all representations are known, and none are mapped to $\perp$). Obviously, the true state of the world has to be an extension of the current state, but predictions about the accuracies of unexplored representations place further restrictions on the set of possible extensions:

$$KN = \{S_i\} \ s.t. \ (S_{true} \in \{S_i\}) \ \wedge \ (S_i \succ S_{current}).$$

Note that the expert is not required to hold a belief about the accuracy of every possible representation. Extensions allow the expert to comment on only those representations of interest, given the current state, and to be ambivalent about all the other representations for which there are no particular expectations.

Domain experts typically find it difficult to make predictions as specific as the exact accuracy expected from a proposed change-of-representation. Instead, a simpler task is to state expectations about the *relative order* for the accuracies of some new representations. That is, given the current state, the expert might able to suggest some representations that will be better, and some that should be worse. For example, given that r_2 was evaluated to have 86% accuracy, the expert might believe that the accuracy achievable with r_3 will be even higher, without being able to give a precise estimate. Recalling that the search state function $S(\cdot)$

maps representations to accuracies, and that the accuracies for all representations are known in S_{true}, we can write this belief as follows: $S_{true}(r_3) > S_{true}(r_2)$. Though such a statement does not entail a unique minimal extension (a specific expected state), it clearly puts a constraint on the possible extensions to the current state. That is, of all the possible ways the true state of the world could be (where the accuracies for all representations are known), only the ones that are plausible are the ones that satisfy the given ordering.

More generally, knowledge can be thought of as a *partial order* (p-order) among representations. A partial order is simply a binary relation $>_p$, or a set of ordered pairs extensionally, which as directed graph would be acyclic:

$$>_p \subset \{(r_i, r_j)\} \text{ where } r_i, r_j \in \mathcal{R}$$

$$(r_i >_p r_j) \rightarrow \neg(r_j >_p r_i) \text{ (consistency)}$$

$$(r_i >_p r_j) \wedge (r_j >_p r_k) \rightarrow (r_i >_p r_k) \text{ (transitivity)}$$

Some of the r_i or r_j may be known in the current state (i.e. $S_{current}(r_j) \neq \perp$), which is common when the expert's beliefs are given as expectations about neighbors of nodes already evaluated. The meaning of such a p-order is that the accuracies of the involved representations are expected to be in their given orders in the true state where all accuracies are known:

$$r_i >_{p_k} r_j \Rightarrow S_{true}(r_i) > S_{true}(r_j).$$

Thus knowledge in our framework is a function K that maps a partial state function into a partial order:

$$K(S_i) = >_{p_i} .$$

This abstract formalism establishes the semantic basis for the role domain knowledge plays in our framework. It formalizes what we mean for domain knowledge to be made "relevant to representation" by giving a specific knowledge representation language for expressing the knowledge. In particular, a theory about the domain must be adapted so that it can compute a partial order on the expected accuracies of unexplored representations, based on those examined so far. This function can be implemented in any way that is convenient for the domain expert (we discuss how to adapt Horn-clause theories below). Such a partial order can be interpreted as expectations about the accuracies of other representations in the true state of the world, which constrain the possible extensions of the partial function encoding the state of the search.

Given this description of the strategy layer, here is how all three components interact. Note that the search state is initialized by evaluating the learner on the examples in their original representation r_1, and this accuracy score $a_1 \in [0,1]$ is the only member in the initial partial function S_1. In a new cycle, the strategy layer is given the partial state function, from which it can compute a p-order $>_{p_i}$ among new representations. The strategy layer is asked to return a *maximal* representation r_i such that there is no representation r_j believed to be better than it in the p-order:

$$\exists_{r_k}(r_i >_{p_i} r_k) \wedge \neg\exists_{r_j}(r_j >_{p_i} r_i).^2$$

The selected representation is passed down to the evaluation component, which applies the representation to a set of examples, transforming them from their original descriptions. The data is randomly split into training and testing sets, and the accuracy of the learner is computed by cross-validation. This accuracy $a_i \in [0,1]$ is returned to the search space component, which updates the current state function by changing the mapping of r_i from $\perp$ to a_i:

$$S_{i+1}(r_x) = S_i(r_x) \text{ except } S_{i+1}(r_i) = a_i.$$

Finally, the process iterates, with the strategy layer computing a new p-order and selecting the next best representation to evaluate, given the new state of the search.

[2]There may be multiple maximal representations, among which an arbitrary choice is made; any more intelligent choice should be encoded in the knowledge to help disambiguate the ordering.

Chapter 4

Implementation and Example

While the framework provides a formalism for incorporating domain knowledge in learning via a partial order among alternative representations, the framework does not define the actual language used to encode the knowledge. Perhaps the most general language is first-order logic, which can be used to encode almost any concepts and does not have any implicit semantics affecting interpretation, allowing the expert to express things as directly as possible. An implementation of our framework could be used to derive maximal representations from such knowledge, but automated deduction from first-order theories is often inefficient. Horn-clauses are a better representation for knowledge because, even though Horn theories are similar to first-order logic, efficient resolution strategies are known. In this chapter, we describe an implementation of the framework for Horn-clauses called CORIL (for "Change-Of-Representation In Learning"). Then we give an extended example, focusing at first on how domain knowledge can be encoded in the formalism of the framework, and then showing how the example could be implemented in CORIL to actually solve the problem.

4.1 Implementation: CORIL

CORIL is based on CLP(R), a constraint logic-programming language (Jaffar and Lassez, 1987). Like PROLOG, CLP(R) can make inferences from Horn-clause theories using SLD resolution (Lloyd, 1988). The requires the domain knowledge to be translated into Horn-clauses, though this format is fairly close in syntax and semantics to first-order logic, making it fairly easy to convert theories either automatically or with some minor manual assistance. CLP(R) extends

PROLOG by adding a special constraint-solving engine for handling arithmetic, which we find to be generally useful in domains to which our framework might be applied.

We will describe how each component of our framework can be implemented in CLP(R). The strategy component requires domain knowledge to specify a partial order among alternative representations. CORIL assumes the partial order is represented by the predicate 'order.' Furthermore, the framework describes the partial order as a function of the current search state. To enable this, CORIL asserts the predicate 'state(R,A)' for each evaluated representation, meaning that A was the accuracy that resulted from evaluating representation R. The domain theory, which must ultimately must make conclusions about the 'order' predicate, may rely on the 'state' predicate to make the search dynamic. Representations themselves may be reasoned about using various syntactic constructions in CLP(R), such as the list notation '$[A|B]$.' Here is a simple example, which says if representation R has been evaluated, and F in an interesting feature, then a good alternative might be to append the feature to the base representation. Extensive domain knowledge might be used to define what are interesting features.

```
order([F|R],R) :-
  interestingFeature(F),
  state(R,A).
```

The implementation of the evaluation component in CORIL requires the specification of ways in which various resources may be supplied. Given a representation, CORIL will apply it to some examples, divide them into training and testing sets, and run an induction algorithm on them to estimate the effect on accuracy. The examples should be given via a three-argument predicate 'example' that specifies a unique identifying name, the class label of the example, and a description of the example. The description must be encoded in a single term, perhaps using the list notation, which differs from other Horn-clause-based systems which allow the description of an example to be spread-out over multiple facts. Here are some examples of cars, as simple feature vectors.

```
example(ex1,pos,[ford,escort,red,4door,1990]).
example(ex2,neg,[VW,rabbit,blue,4door,1980]).
```

To apply a representation to a given example, CORIL calls a predicate 'transform,' which must be defined by the user. The 'transform' predicate has three arguments: an input example, an alternative representation, and the output example that results from changing the example into the new representation. The expert must add some rules that specify what the alternative representations in the 'order' relation actually mean. Their effect on examples should be a consequence of the background knowledge. Here is an example of a transformation, which assumes that examples are lists of features and alternative representations are given by simple subsets of those features. It iterates down the list of features, picks the appropriate value out of the input vector, and appends it to the output.

```
transform(E,[],[]).
transform(E1,[F|R],[G|E2]) :-
  transform(E1,R,E2),
  initialRepresentation(Features),
  nth(N,Features,F),
  nth(N,E1,G).
```

With transformed examples, the evaluation routine must call the induction algorithm chosen by the expert. CORIL takes advantage of CLP(R)'s built-in 'oracle' predicate to make calls to an external binary executable program. Thus the expert may simply name a separate learning program, and it will be invoked by the evaluation routine. The learning program must conform to a special interface to interact with CORIL. CORIL will write the transformed descriptions of examples into the learning program's standard input, indicating which examples are for training and which for testing, and then wait for the algorithm to return a floating point performance score (typically inductive accuracy). The protocol is as follows: the training examples are given in sequence, alternating the class label on one line and description on the next. Then the message "doneTraining" is sent. Then the testing examples are given, with class labels and descriptions on alternating lines. And finally, "doneTesting" is sent:

```
class 1
description 1
class 2
description 2
...
doneTraining
```

```
class i
description i+1
...
doneTesting
```

Here is the CLP(R) code for calling the learning program and communicating with it. The routine needs to know the name of the binary, which the user supplies via the fact 'learner(Name).' The input consists of a list of training examples and a list of testing examples, each as pairs of class labels and descriptions, which have already been transformed into the representation of interest.

```
learn(Train,Test,Acc) :-
  learner(Name),              % user-specified
  oracle(input,output,Name), % invoke binary, get pipes
  see(output), % redirect stdin of CLP(R) to binary's stdout
  tell(input), % redirect stdout of CLP(R) to binary's stdin
  writeExamples(Train),
  write(doneTraining),nl,
  writeExamples(Test),
  write(doneTesting),nl,
  read(Acc),  % get float printed by binary to stdout
  see(user),  % reset stdin for CLP(R)
  tell(user). % reset stdout for CLP(R)

writeExamples([]).
writeExamples([[Class,Description]|Rest]) :-
  write(Class),nl,
  write(Description),nl,
  writeExamples(Rest).
```

Here is the CLP(R) code for the evaluation routine, which calls the predicates defined above for transforming examples and invoking the learning program. The additional procedures here take care of cross-validation by manipulating subsets of the available examples.

```
eval(Repr,Acc) :-
  gatherExamples([],ExList),
  applyTransformation(ExList1,Repr,ExList2),
  splitIntoFolds(ExList2,10,Folds),
  evalFolds([0,1,2,3,4,5,6,7,8,9],Folds,AccList),
  average(AccList,Acc).

% builds list of names of examples
gatherExamples(List,Result) :-
  example(Ex,_,_),
  not member(Ex,List),
  gatherExamples([Ex|List],Result).
gatherExamples(List,List).

% for each example name, replace with class and
%   transformed description
applyTransformation([],_,[]).
applyTransformation([Ex1|List1],Repr,[[Class|Trans]|List2]) :-
  example(Ex1,Class,Descr),
  transform(Descr,Repr,Trans),
  applyTransformation(List1,Repr,List2).

% divide set randomly into N subsets, roughly balanced
splitIntoFolds(Set,N,Folds) :-
  listOfEmptyLists(N,List), % returns like [[],[],[]] for N=3
  splitIntoFolds2(Set,N,List,Folds).
splitIntoFolds2([],N,List,List).
splitIntoFolds2([First|Rest],N,List,Folds) :-
  rand(X),
  Y=round(N*X),
  nth(Y,List,Subset),
```

```
  replaceNth(N,List,[First|Subset],NewList),
  splitIntoFolds2(Rest,N,NewList,Folds).

% call evalFold for each fold number in list
evalFolds([],Folds,[]).
evalFolds([First|Rest],Folds,[A|B]) :-
  evalFold(First,Folds,A),
  evalFolds(Rest,Folds,B).

% divide data into training and testing sets by removing subset
%   for requested fold; invoke learner to get accuracy score
evalFold(N,Folds,Acc) :-
  nth(N,Folds,Test),
  removeNth(N,Folds,Train1),
  flatten(Train1,Train).
  learn(Train,Test,Acc).
```

The search component is what drives the framework by mediating the interaction between the strategy component and the evaluation component, iteratively selecting a suggested representation and evaluating it. The search component queries the strategy component for a maximal representation, which is an element in the 'order' relation such that there is no better element. CORIL implements this via a rule for 'maximal(R),' which it tries to prove as a goal. A solution to this goal constitutes one of the best representations implied by the domain knowledge for the current search state. The search routine calls this goal, passes the selected representation to the evaluation component, and caches the resulting accuracy in the 'state' predicate using the 'assert' function built into CLP(R). CORIL implements a simple stopping procedure that halts after a specified number of iterations, though other implementations could choose to stop only after a certain amount of improvement has been achieved, etc.

```
maximal(R) :-
  order(R,_),
  not order(_,R).
```

```
search(0,Repr) :-
  bestRepr(Repr).
search(Iters,Repr) :-
  Iters>0,
  maximal(R),
  eval(R,A),
  assert(state(R,A)),
  search(Iters-1,Repr).

% find representation with highest acc in 'state' predicate
bestRepr(Repr) :- bestRepr2([],0,Repr).
bestRepr2(A,B,C) :-
  state(D,E),
  E>B,
  bestRepr2(D,E,C).
bestRepr2(A,B,A).
```

Here is a summary of the things a user needs to supply to CORIL:

1. examples (in the 'example' predicate)
2. name of the learning program (a binary that uses the special communication protocol via stdin/stdout)
3. domain knowledge in Horn clauses
4. rules for the 'order' relation
5. rules for the 'transform' function

4.2 Example

We now give an example of using the formalism of the framework to encode some background knowledge in a particular domain to help decide which alternative representations to explore during search. Imagine a physician who is trying to getting a learning algorithm like ID3 to recognize a particular kind of infection from the records of patients in a clinic. Suppose the records contain measurements for the body temperature of each patient. The physician knows that high temperatures can indicate a fever, which can be a sign of infection. Thus the physician decides that it might be useful to the learner the patients were divided into two subsets depending on whether their body temperature was above some cutoff. However, the physician is uncertain where to draw the exact cutoff for fevers. Thus he can use our framework to search for an optimum. Our goal is to show how to encode the physician's background knowledge in our formalism such that it generates suggestions for alternative representations.

We will encode it in first-order logic (FOL), since this has an simple and agreed-upon semantics (Genesereth and Nilsson, 1987). Recall that knowledge in our framework is formally defined as a mapping from partial state function to a partial order among other representations based on their expected effects on accuracy. To simulate this, we will use some special 'input' and 'output' predicates. The predicate $state(r, a)$ will be asserted as a fact for each representation r that has been evaluated so far in the search, with a being its observed accuracy. The theory will have to be adapted to conclude $order(r_1, r_2)$ for each pair of alternative representations such that $acc(r_1) > acc(r_2)$. The partial order, as a mathematical relation, will arise from the model-theoretic interpretation of 'order' predicate (Genesereth and Nilsson, 1987).

We begin by formalizing the intuition that fevers are sign of infection, so discretizing temperature at a cutoff slightly above normal body temperature might help learning. First, infection is our target concept, and infections are usually associated with bacteria, which produce toxins in the blood. In the physician's naive theory of human physiology, the body responds to toxins by a complicated mechanism that raises the body temperature as a side-effect. By this chain of reasoning, then, a fever is a sign of an infection.

$$targetConcept(infection)$$
$$present(infection) \rightarrow present(bacteria)$$
$$present(bacteria) \rightarrow present(toxinsInBlood)$$
$$present(toxinsInBlood) \rightarrow present(raisedBodyTemp)$$

$$present(raisedBodyTemp) \rightarrow present(fever)$$

To refine this theory, the physician must give a definition for "fever," which can be done indirectly through defining "raised body temperature." A raised body temperature is simply a temperature that is higher than normal human body temperature. The average normal body temperature is known to be $98.6°F$. However, body temperatures naturally vary, so being significantly above normal means being greater by at least one standard deviation. Thus fever might be defined as a new attribute for each patient which is true if the patient's temperature is greater than the average normal body temperature plus one standard deviation, and false if the patient's temperature is below this cutoff. The physician might not know the standard deviation offhand, but might be able to guess that it is between 1 and 3 degrees.

$$\begin{aligned}
&present(raisedBodyTemp) \ \wedge \ normalBodyTempAverage(A) \\
&\quad \wedge \ normalBodyTempStdDev(B) \ \wedge \ value(temp, V) \rightarrow (V > A + B) \\
&normalBodyTempAverage(98.6) \\
&normalBodyTempStdDev(X) \rightarrow (1 < X < 3)
\end{aligned}$$

This domain knowledge must be made relevant to representation, and according to our formalism, this connection should be made via the 'order' relation, which gives expected relative improvements among some new representations. A general piece of knowledge which can be applied to many domains is that, when one thing is a sign of another, then a feature that determines a value for the first thing for each example might help a learner recognize the second thing. In more concrete terms, if you append the new feature to the initial representation, then this alternative representation should allow the learner to get higher accuracy, which finally can be linked to the 'order' predicate. The domain expert might not know this rule, but a knowledge engineer could supply it, or it might even be built into an implemented system.

$$\begin{aligned}
&targetConcept(A) \ \wedge \ (present(A) \rightarrow (B > C)) \\
&\quad \rightarrow usefulForRecognizing(A, construct(B, '>', C)) \\
&usefulForRecognizing(A, F) \ \wedge \ initialRepresentation(R_0) \\
&\quad \rightarrow order(append(F, R_0), R_0)
\end{aligned}$$

The theory developed so far can be used to suggest certain alternative representations for our search. In particular, it generates a whole set of potential changes to the initial representation, each of which appends a unique discretization of temperature based on a different cutoff. These representations are individual neighbors of the start node in the search space. The solution set for maximal representations according to the 'order' predicate is constrained so that the cutoff lies between 99.6 and 101.6 (average body temperature plus one standard deviation, which is not precisely known). This knowledge implies a simple search procedure. With each iteration of search, a random cutoff in this range would be chosen, the corresponding feature would be constructed and appended to the initial representation, and the new representation would be evaluated to see if it improves the accuracy of the learner.

We have shown how background knowledge can be transformed into a search procedure for exploring alternative representations. However, the search procedure is rather simplistic; it does not even depend on the state of the search (i.e. the representations evaluated so far). Intuitively, it seems that there should be some more intelligent search procedure that exploits some additional pieces of information about the domain. For example, the physician might believe that the utility of these discretizations of temperature for learning to recognize infections is likely to be a single-peaked (i.e. *unimodal*) function of where the cutoff is drawn: utility will increase to maximum and then decrease. This constraint can be used to gradually restrict the range of values for random selection as search progresses. We will show where this unimodal assumption comes from, in terms of the physician's knowledge, and we will show how it can dynamically improve the suggestions for alternative representations, hence making the search more efficient.

The unimodal assumption is grounded in the expert's belief that there are there are two distinct distributions of body temperatures: one for healthy people and one for sick people. The expert may assume that each distribution is normal (Gaussian), with the mean temperature for healthy people significantly below that for people with infections (because of the above explanation about fevers). For two normal distributions (bell curves) to be significantly separated, the differences between their means should be at least on the order of the sum of their standard deviations.

$$\neg present(infection) \ \wedge \ value(temp, V) \rightarrow normalDistribution(V, \mu_h, \sigma_h)$$
$$present(infection) \wedge value(temp, V) \rightarrow normalDistribution(V, \mu_s, \sigma_s)$$

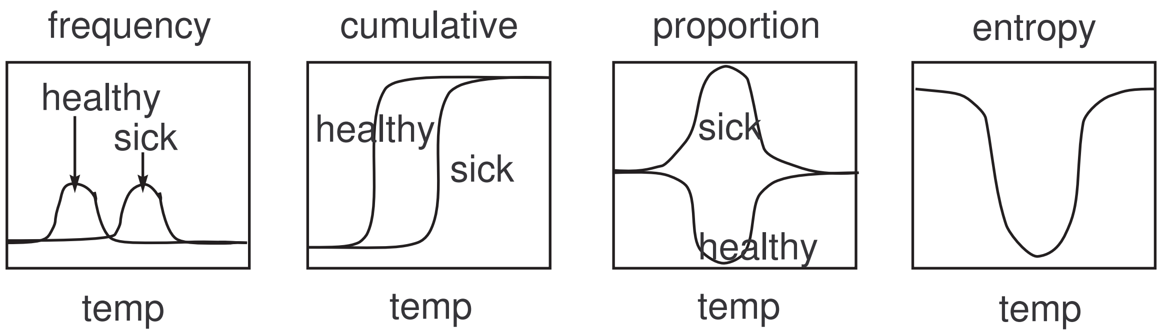

Figure 4.1: Qualitative demonstration that when one variable divides a population into two separate peaks, the entropy gained when the variable is discretized at various cutoffs in this range will have the shape of a valley. The cumulative distributions will be sigmoids that make their transitions at two different locations. Thus the relative proportions of the two sub-populations (sick versus healthy in this case) among instances with a value higher than the cutoff will be around 50% except when the cutoff is between the two peaks. By straightforward calculations, these proportions will lead an entropy curve that dips to a minimum in the middle.

$(\mu_s - \sigma_s > \mu_h + \sigma_h)$

When the sub-populations for each state of a Boolean condition like infection fall into separate bell-curves with respect to a numeric variable like temperature, the entropy gained by dividing the population at various cutoffs will look like a valley. This can be seen qualitatively as follows (see Figure 9). First, consider that, for two normal curves, the cumulative distributions for each will be sigmoids that go from 0% to 100% in two different locations (where the respective means are). Using these two cumulative distributions, one can deduce that the relative proportion of patients with the infection in the population with body temperatures above the cutoff will start out as around 50%, rise up close to 100%, and then drop to 50% again. When the cutoff is below the first bell-curve, both sub-populations are largely above the cutoff (proportion infected=50%); between the curves, only people with the infection have temperatures higher than the cutoff (proportion infected=100%); and above the second bell-curve, neither population has much presence, so the proportion is relatively balanced again. We could encode this argument using some qualitative rules for reasoning about graphs, or perhaps we could encode a more formal mathematical derivation. However, for conciseness, we state the observation as an axiom.

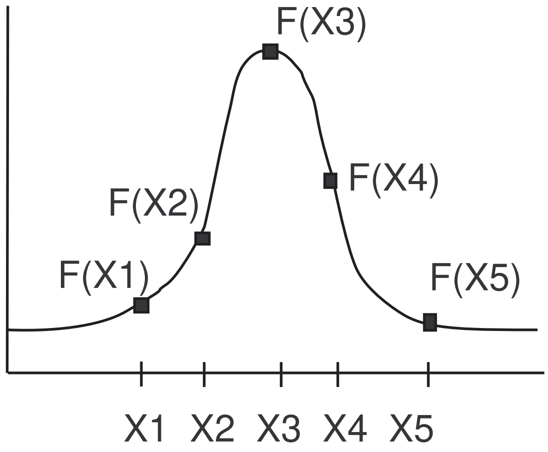

Figure 4.2: Deriving constraints on the values of a function, given that it has the shape of a peak. There exists a value (X3) at which the function reaches its maximum. The function is monotonic on either side of this point, which can determine which value of the function at two points (e.g. X1 and X2) is greater based on their order.

$$
\begin{aligned}
&((present(A)\ \wedge\ value(X,V)) \rightarrow normalDistribution(V,\mu_a,\sigma_a)) \\
&\quad \wedge ((\neg present(A)\ \wedge\ value(X,V)) \rightarrow normalDistribution(V,\mu_b,\sigma_b)) \\
&\quad \wedge (\mu_a - \mu_b > \sigma_a + \sigma_b) \\
&\quad \rightarrow shapeOfCurve(func(entropy(A, construct(X,'>',Y)),Y), valley)
\end{aligned}
$$

To say that a curve looks like a valley is meaningless without other axioms that constrain its interpretation. One simple rule might be that, if a curve looks like a valley, then the complement of that function (subtracted from a constant) would have single peak. The complement of entropy is equivalent to the utility of a variable for recognizing the target concept, since low entropy means the one variable separates sub-populations of the other well. If the entropy function looks like a valley, then the utility function will look like a peak, which completes the explanation of the unimodal assumption. To formally define this property, if a function F has a single peak, then there exists a value X such that (1) F is a maximum at X, and (2) F is monotonic on either side of X. That is, the order of the values of the function at two points on either side can be determined by the order of the points themselves.

$$
\begin{aligned}
&shapeOfCurve(func(F,C), valley) \wedge complement(F,G) \\
&\quad \rightarrow shapeOfCurve(func(G,C), peak) \\
&complement(entropy(A,B), utilityForLearning(A,B)) \\
&shapeOfCurve(func(A,X), peak) \rightarrow \exists_{X_3}\ value(A, X_3, A_3) \\
&\quad \wedge (\forall_{X_1,X_2}\ (X_1 < X_2 < X_3) \wedge value(A, X_1, A_1) \wedge value(A, X_2, A_2) \\
&\qquad \rightarrow (A_1 < A_2 < A_3)) \\
&\quad \wedge (\forall_{X_4,X_5}\ (X_3 < X_4 < X_5) \wedge value(A, X_4, A_4) \wedge value(A, X_5, A_5) \\
&\qquad \rightarrow (A_3 > A_4 > A_5))
\end{aligned}
$$

This property can be used to derive some new constraints on which cutoffs are better than others. In particular, if we know that the function for the utility for learning how to recognize infections has a single peak with respect to various cutoffs for temperature, then we can use this to infer some ranges where the maximum is not likely to be, given some known points on this curve (i.e. for representations already evaluated). If three points in a row have increasing utility (e.g. $x_1 < x_2 < x_3$ in Figure 10), then every cutoff below the central point can be expected to have less utility than those between the central and upper points. Similarly, if the central point of three points in a row is greater than the other two (e.g. $x_2 < x_3 < x_4$ in Figure 10), then it can be proven that any cutoff outside each endpoints should have less utility than cutoffs between that endpoint and the central point. These lemmas can be derived from the unimodal assumption.

$$
\begin{aligned}
&(value(f(A, X_1, A_1)) \wedge value(f(A, X_2, A_2)) \wedge value(f(A, X_3, A_3)) \\
&\qquad \wedge (X_1 < X_2 < X_3) \wedge (A_1 > A_2 > A_3)) \\
&\quad \rightarrow (\forall_{X_4,X_5}\ (X_1 < X_4 < X_2) \wedge (X_3 < X_5) \wedge value(f(A, X_4, A_4)) \\
&\qquad \wedge value(f(A, X_5, A_5)) \rightarrow (A_4 > A_5) \\
&(value(f(A, X_1, A_1)) \wedge value(f(A, X_2, A_2)) \wedge value(f(A, X_3, A_3)) \\
&\qquad \wedge (X_1 < X_2 < X_3) \wedge (A_1 < A_2 > A_3)) \\
&\quad \rightarrow (\forall_{X_4}(X_4 < X_1) \wedge value(f(A, X_4, A_4)) \rightarrow (A_4 < A_1))
\end{aligned}
$$

$$\wedge\ (\forall_{X_5}(X_3 < X_5)\ \wedge\ value(f(A, X_5, A_5)) \rightarrow (A_3 < A_5))$$

These new constraints can lead directly to a more focused search, which effectively becomes a hill-climbing procedure. We will add some new rules that augment the partial order by making some new links among the suggested representations themselves (which only dominated the initial representation previously). This is done with a simple rule that says, if two representations that each contain a discretization of temperature are based on cutoffs that are known to have utilities for learning in some particular order, then put this pair directly in the 'order' predicate. Hence when the theory is solved to find a maximal representation, these new constraints will prevent cutoffs in certain subranges from being considered.

$$\begin{aligned}
&targetConcept(A)\ \wedge\ initialRepresentation(R_0)\\
&\quad \wedge\ value(utilityForLearning(A, F_1), C_1, A_1)\\
&\quad \wedge\ value(utilityForLearning(A, F_2), C_2, A_2)\\
&\quad \wedge\ (A_1 > A_2) \rightarrow order(append(F_1, R_0), append(F_2, R_0))
\end{aligned}$$

As more and more representations accumulate on the slopes of the utility curve during search, the constraints of the unimodal assumption will begin to rule out the edges of the range. The utility of a representation is explicitly equated with its effect on the accuracy of learning. Individual representations that have already been evaluated, along with their observed accuracies, are given extensionally via the 'state' predicate. These representations are deconstructed to determine their relationship between the temperature cutoff they used and the accuracy that resulted, and hence their utility for learning. The constraints will then automatically make inferences about the utilities for cutoffs in between these discrete points. The range of reasonable cutoffs will gradually be restricted, providing a more selective guess, and thus homing in on the identity of the optimal cutoff.

$$\begin{aligned}
&targetConcept(X)\ \wedge\ initialRepresentation(R_0)\\
&\quad \wedge\ state(append(construct(F,'>', C), R_0), Acc)\\
&\quad \rightarrow value(utilityForLearning(X, construct(F,'>', C)), C, Acc)
\end{aligned}$$

4.3 Application of CORIL to the Example

We have described the knowledge in this domain that motivates changing the representation of patient records by discretizing the body-temperature variable, based on the association between fevers and infections. This knowledge was adapted to the framework by adding rules for drawing conclusions about which representations are expected to improve learning, via the 'order' predicate. However, this knowledge was given in first-order logic (FOL), and was not directly executable. Here we show how the knowledge can be translated into Horn-clauses for use in CORIL, an implementation of our framework based on CLP(R) that could actually be used to improve prediction in this domain.

The first set of knowledge involved establishing the connection between infections, the target concept, and raised body temperatures, for which we have a concrete definition. Note that the earlier knowledge cannot be directly translated into Horn-clauses because it contains some disjunction. Some other implementation of our framework might be able to make inferences directly from these statements in FOL, though it would have to have a sophisticated theorem-proving technique to be efficient. To use CORIL, we must translate the knowledge into a Horn-clause theory that reasonably reflects the content of the original knowledge, resulting in a slightly different appearance.

```
implies(infection,bacteria).
implies(bacteria,toxinsInBlood).
implies(toxinsInBlood,raisedBodyTemp).
% this is necessary to achieve transitivity
implies(A,C) :- implies(A,B),implies(B,C).

definition(raisedBodyTemp,higherThanNormal(temp)).

% concrete feature for 'higherThanNormal' for arbitrary
%   variable X, based knowledge of the distribution for X
% the function terms 'gt' and 'lt' mean greater/less than
% we use the term '[A,B]' often to mean a range real nums
feature(higherThanNormal(X),gt(X,[A+B,A+C])) :-
```

```
  average(X,A),
  stdev(X,[A,B]).

average(temp,A) :- normalBodyTemp(A).
stdev(temp,B) :- stdevBodyTemp(B).

normalBodyTemp(98.6).
stdevBodyTemp([1,3]). % std. dev. between 1 and 3
```

Now we add some rules that make explicit references to representation. These rules state that appending a feature believed to be useful for recognizing the target concept would improve the representation for learning. In conjunction with the previous rules about the relationship between infections and raised body temperatures, this knowledge can make the first entry in the 'order' relation. Here we need to define the target concept and the initial representation, which in this domain is a list of features.

```
goodForRecognizing(I,F) :-
  implies(I,B),
  definition(B,D),
  feature(D,F).

order(append(F,R),R) :-
  targetConcept(I),
  initialRepresenation(R),
  goodForRecognizing(I,F).

targetConcept(infection).
% format for examples, including body temperature (temp)
%   and red and white blood cell counts (redBCs, whiteBCs)
initialRepresentation([name,age,sex,rage,temp,redBCs,whiteBCs]).
```

At this point, we can conclude that representations in which a discretized feature for temperature is appended might be better than the initial representation. Now we need to provide some rules for CORIL to transform examples into these new representations (during the **eval** routine). Representations will be pre-processed into lists of features (including the newly generated representations, such as 'append(gt(temp,100),[name,age...]),' which is easily translated into '[gt(temp,100),name,age...].'). Because our translation has used an explicit representation of ranges of variables, solutions really correspond to sets of equivalent representations, one of which is picked randomly prior to calling the transformation routine.

```
preProcess(append(gt(T,[A,B]),R),[gt(T,C)|R]) :-
  C=A+rand(X)*(B-A).

% an example E1 transforms into an empty list for
%   representation with no features
transform(E1,[],[]).

% if first feature is in original representation,
%   pull value out of same position in example
transform(E1,[F|R1],[G|E2]) :-
  transform(E1,R1,E2),
  initialRepresentation(R0),
  nth(N,R0,F),
  nth(N,E1,G).

% for discretized features, look up base value in original
%   example, compare to cutoff, append TRUE if succeeds
transform(E1,[gt(F,C)|R1],[True|E2]) :-
  transform(E1,R1,E2),
  initialRepresentation(R0),
  nth(N,R0,F),
  nth(N,E1,V),
  V>C.

% if value for discretized feature does not exceed cutoff,
```

```
%   append FALSE to output example
transform(E1,[gt(F,C)|R1],[False|E2]) :-
  transform(E1,R1,E2),
  initialRepresentation(R0),
  nth(N,R0,F),
  nth(N,E1,V),
  V<=C.
```

In addition to the knowledge that discretizing temperature in the inferred range should help learning, we need to encode the rules for shifting preferences among sub-ranges as search progresses. These constraints are derived from the belief that the utility of discretizing temperature should be a unimodal (single-peaked) function of the cutoff.

```
abnormalCondition(infection).
% if X is abnormal, then no(X) is a normal condition
% so distributions are based on normal averages, etc.
implies(no(X),lt(T,[A+B,A+C])) :-
  abnormalCondition(infection),
  average(T,A),
  stdev(T,[B,C]).

% if two populations I and J have values for T that are gt and lt C,
%   their peaks will be separate when graphed against T
separatePeaks(I,J,T) :-
  goodForRecognizing(I,gt(T,C)),
  implies(J,lt(T,C)).

% if the peaks for the presence and absence of a
%   condition are separated on axis T, the entropy
%   gained by discretizing T at various cutoffs will
%   dip in the middle, where it separates the peaks
shapeOfCurve(entropy(I,T),valley) :-
```

```
  separatePeaks(I,no(I),T).

% qualitative reasoning about geometry
shapeOfCurve(F,peak) :-
  shapeOfCurve(G,valley),
  complement(F,G).

complementFunc(utilityForLearning(I,T),entropy(I,T)).
```

As points are evaluated, they place constraints on which sub-ranges probably contain the best alternatives. This is knowledge connects logic about values of unimodal functions directly to the 'order' predicate. We use accuracies in the 'state' predicate to provide known points on the 'utilityForLearning' curve, which is the basis for making the search dynamically depend on previous results.

```
known(utilityForLearning(I,T),C,Acc) :-
  state(append(gt(T,C),R),Acc).

% if F is a single-peaked curve, and C1, C2, and C3 are three
%   points whose values are known to be V1, V2, and V3,
%   and if the points are increasing (C1<C2<C3, V1<V2<V3)
%   any points in the range below the midpoint [C1,C2] should
%   have lower values than those in the range above [C2,C3]
dominantRange(F,[C2,C3],[C1,C2]) :-
  shapeOfCurve(F,peak),
  known(F,C1,V1),
  known(F,C2,V2),
  known(F,C3,V3),
  C1<C2,C2<C3,
  V1<V2,V2<V3.

% symmetric constraint for points on other side of peak
```

```
dominantRange(F,[C1,C2],[C2,C3]) :-
  shapeOfCurve(F,peak),
  known(F,C1,V1),
  known(F,C2,V2),
  known(F,C3,V3),
  C1<C2,C2<C3,
  V1>V2,V2>V3.

order(append(gt(T,A),R),append(gt(T,B),R)) :-
  initialRepresentation(R),
  targetConcept(I),
  dominantRange(utilityForLearning(I,T),A,B).
```

Because the constraint solver in CLP(R) is not quite strong enough, we must explicitly add rules for reasoning about ranges, particularly when they overlap.

```
% get transitivity when ranges overlap
order(U,append(gt(T,[C1,C2]),R)) :-
  order(U,append(gt(T,[C7,C8]),R)),
  order(append(gt(T,[C1,C2]),R),W),
  C7<X,X<C8,
  C1<X,X<C2.

% the following rules split apart ranges when they
%   are only partially dominated
order(append(gt(T,[C8,C2]),R),W) :-
  order(U,append(gt(T,[C7,C8]),R)),
  order(append(gt(T,[C1,C2]),R),W),
  C7<C1,C1<C8,C8<C2.

order(append(gt(T,[C1,C7]),R),W) :-
  order(U,append(gt(T,[C7,C8]),R)),
```

```
  order(append(gt(T,[C1,C2]),R),W),
  C1<C7,C7<C2,C2<C8.

order(append(gt(T,[C8,C2]),R),W) :-
  order(U,append(gt(T,[C7,C8]),R)),
  order(append(gt(T,[C1,C2]),R),W),
  C1<C7,C8<C2.

order(append(gt(T,[C1,C7]),R),W) :-
  order(U,append(gt(T,[C7,C8]),R)),
  order(append(gt(T,[C1,C2]),R),W),
  C1<C7,C8<C2.
```

Chapter 5

Protein Structure Prediction and Homology Modeling

In the previous chapters, we have introduced a new framework for understanding change-of-representation in machine learning. To demonstrate our framework, we will apply it to a difficult, real-world domain: protein structure prediction, an important but very difficult problem in biology. Proteins are macromolecules produced by cells that serve a wide range of biological functions (Stryer, 1988). Proteins are linear polymers of smaller compounds called amino acids which fold into complex three-dimensional structures soon after synthesis. Currently, only the amino acid sequences are known for many proteins (Watson, 1990). However, it is knowledge of the structure of a protein that reveals the deepest insights about its function. Since it is generally difficult to determine the structure of a protein using laboratory methods, computational methods are needed to predict the 3D structure of a protein from its amino acid sequence. This capability could benefit many fields, from medical research (e.g. understanding disease pathways) to chemical engineering (e.g. designing more effective catalysts). The best method to date for predicting protein structures, homology modeling, has only limited accuracy. We will describe the implementation of a learning system based on our framework that improves its ability to predict protein structures based on an expert's advice (i.e. biophysics knowledge).

5.1 The Structure of Proteins

The goal of protein structure prediction is to predict the three-dimensional conformation of a protein from its amino acid sequence. This folded structure is the culmination of a synthesis process of many steps in the cell (Alberts *et al.*, 1983). The instructions for the protein sequence originate as a gene in the DNA, located in the nucleus. DNA encodes information as a string over a 4-symbol alphabet (the nucleotides: adenine, guanine, cytosine, and thymine). During transcription, individual genes are copied into mRNA sequences, which use a parallel 4-letter code. The mRNA transcripts may undergo some processing, such as splicing and poly-adenylation, and eventually migrate to another cellular structure called the ribosome. At the ribosome, the mRNA sequence is translated into a protein sequence by concatenating amino acids into chains typically 100 to 1000 long. There are 20 amino acids, or *residues*, used in all biological systems, each consisting of a common component that becomes a segment of the backbone and a unique side-chain (see Figure 5.1). After translation, the protein folds into a three-dimensional shape, is occasionally modified, and is finally transported to its destination in the cellular environment.

According to the central dogma of molecular biology, the synthesis of a protein is a deterministic process (Alberts *et al.*, 1983). During transcription, each nucleotide in the DNA sequence for a gene gets copied into a corresponding nucleotide in the mRNA transcript. During translation, the mRNA sequence is read in non-overlapping triplets (codons) to derive the sequence of amino acids for synthesizing the protein. The genetic code is a universal many-to-one mapping of the 64 possible nucleotide triplets into the 20 amino acids (Schulz and Schirmer, 1979).

Once the amino acid sequence of a protein has been assembled, it usually takes between several milliseconds and several seconds to fold (Baldwin, 1989). Although the pathways of folding are not well-understood, it is postulated that proteins fold into structures of globally minimum energy (McCammon and Harvey, 1987). White (1961) showed that, after denaturation, proteins can often spontaneously re-fold in the absence of all other cellular components, proving that the amino acid sequence alone determines the structure of a protein. While proteins naturally flex due to Brownian motion (Cooper, 1976), and might undergo slight structural adjustments

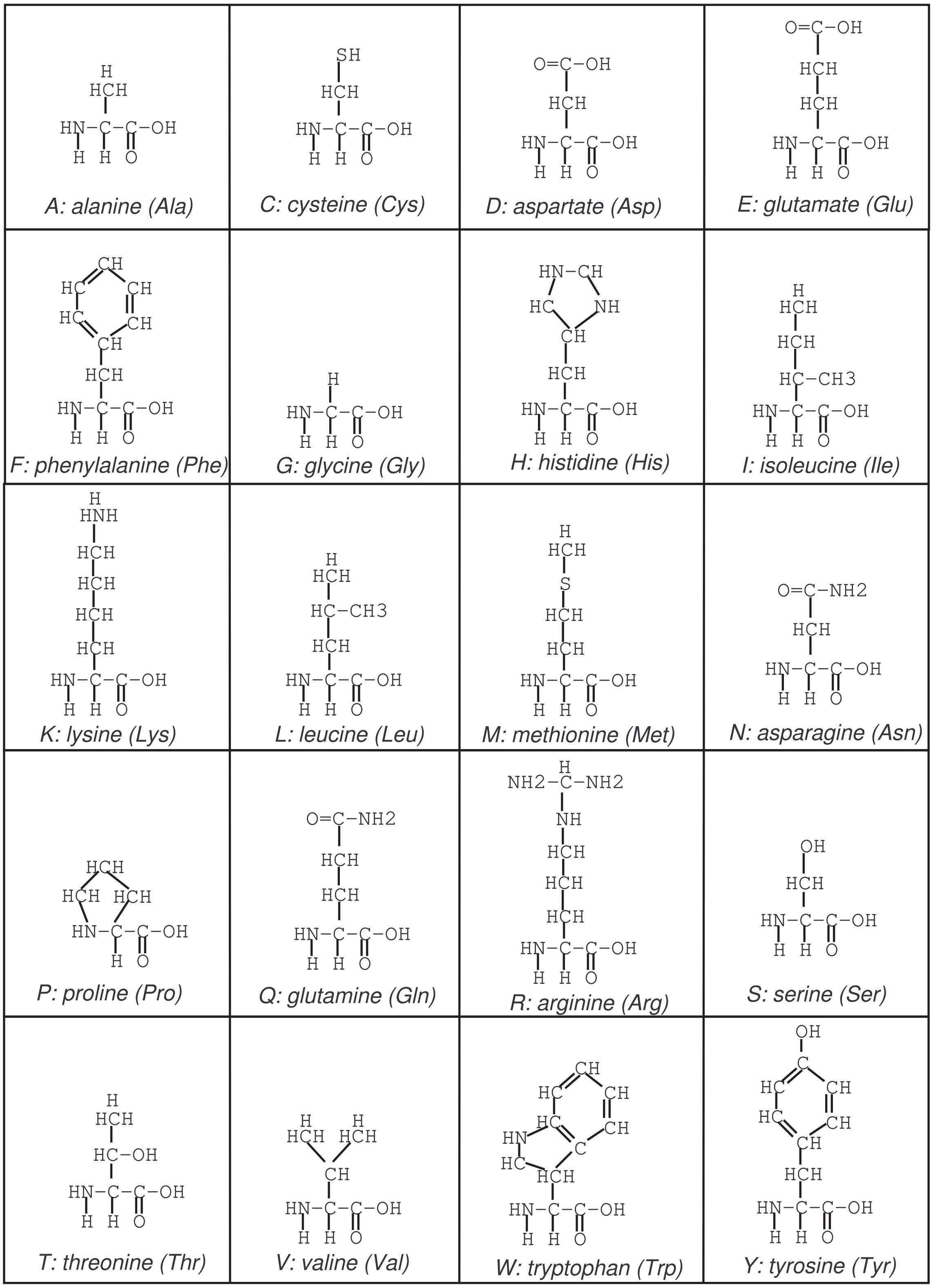

Figure 5.1: The chemical structures of the 20 amino acids, along with their one-letter codes, full names, and 3-letter abbreviations.

during catalysis (Koshland, 1958), these variations are much insignificant compared to the differences in structure between different proteins (Sander and Schneider, 1991).

5.2 Levels of Structural Description

Protein structures can be described at various levels of detail. The lowest level of description is the *primary structure*, which consists of just the names of the amino acids in the sequence. At a higher level, most proteins are said to be globular because they fold into roughly spherical shapes with a diameter on the order of 100 angstroms (although there are exceptions, such as the rod-like collagen molecule; Schulz and Schirmer, 1979). However, internally, each protein has a unique structure determined by the path of the backbone. The trace of backbone atoms contributed by each amino acid forms a super-structure around which the side-chains pack, filling up the volume and generally leaving no cavities (Ponder and Richards, 1987).

Certain sub-structures within proteins have been observed as recurrent patterns in the meander of the backbone (Levitt and Chothia, 1976). There are two main classes of sub-structures, also called *secondary structures*: alpha-helices and beta-sheets. An alpha-helix is a short stretch of about 20 amino acids wound in a tight coil with about 3.5 residues per turn (Srinivasan, 1976). A beta-sheet is composed of two to eight extended stretches of amino acids (called beta-strands, about six residues long on average) aligned side-by-side (Sternberg and Thornton, 1977). In addition to alpha-helices and beta-sheets, certain kinds of turns (4-5 residues long) have been categorized (Thornton and Thornton, 1990). All other sections of backbone are typically labeled as "random coil."

Secondary structure is a global property that emerges from local phenomena. Secondary structures arise from repeating patterns in the twist of the backbone at each amino acid. The peptide bond allows two degrees of rotational freedom, called the phi and psi angles. In a survey of thousands of amino acids in many different proteins, it has been observed that these backbone angles are generally found in two main configurations, primarily due to steric constraints (Ramachandran and Sasisekharan, 1968). Repetitions of $(-60, -60)$ form an alpha-helix, and repetitions of $(-90, +120)$ form a beta-strand. Hydrogen-bonding patterns between backbone atoms have also been used to define secondary structures at the local level, such as between loops in an alpha-helix and between strands in a beta-sheet (Kabsch and Sander, 1983).

At the highest level of description is *tertiary structure*, which refers to the global arrangement of secondary structures, also called the *fold* of the protein (Richardson, 1981). While the structure of a protein can be precisely described using atomic coordinates, it is more convenient to specify the relative locations and orientations of the secondary-structure components. This approach abstracts away insignificant details about atom positions between proteins that are functionally equivalent. Many rules governing the packing of secondary structures are known, such as the propensity for beta-sheets to lie in the core of a protein (Schulz and Schirmer, 1979), and the restriction of contacts between helices to specific angles (Lasters, 1990). Examples of tertiary structures include helix-bundles (four parallel alpha-helices) and beta-barrels (a beta-sheet closed into a cylinder, with 8 alpha-helices packed around the outside; Chothia, 1988).

5.3 Factors Affecting Protein Structure

A variety of factors affect the structure into which a protein will fold (Dill, 1990). The forces involved can be broadly classified as 'entropic' or 'enthalpic.' Entropic forces are mainly concerned with the interaction between the protein molecule and the surrounding solvent molecules, such as water and salt ions. Water molecules tend to form a vast, loose network of hydrogen bonds. Many amino acids are hydrophobic (non-polar) and disrupt the hydrogen-bonding network in the solution surrounding a protein. In response, the water molecules adjacent to a hydrophobic residue will adjust their network to recover most of the hydrogen bonds among themselves. However, this forced organization limits the mobility of the water molecules, decreasing entropy. So entropy favors the packing of hydrophobic residues in the core of a protein, with a roughly spherical surface (to minimize surface area) containing the more polar or ionic residues (Tanford, 1973). This "hydrophobic effect" explains why beta-sheets, which are usually found in the core, generally contain hydrophobic residues, while alpha-helices, which often pack with one side covering the core and one side facing the solvent, are typically amphipathic (alternating hydrophobic and hydrophilic residues).

The other major factors involved in determining the structure of a protein are enthalpic, relating to the free energy of the folded structure. Aside from the covalent bonds, there are many non-covalent interactions which contribute to the stability of a protein (Schulz and Schirmer, 1979). Salt bridges between ionic residues with opposite charges are rare but strong. Polar interactions are slightly more common, since many of the atoms in many of the amino acids carry partial charges. The most frequent polar interaction is hydrogen-bonding, which contributes highly to enthalpy. Each residue contains both a donor and acceptor atom in the backbone, and many residues have additional atoms in the side-chain that can participate in hydrogen bonds. Some non-polar residues can induce polar interactions via resonance, for example between the conjugated rings of aromatic residues. Similarly, hydrophobic side-chains in general are weakly attracted to each other due to the weak Van der Waals force. The free energy gained by satisfying as many of these non-covalent interactions as possible (especially the stronger ones) is a driving force in protein folding.

5.4 Structure Determination and Prediction

Various methods exist for obtaining structural information about a protein. Empirical methods use material samples prepared in a laboratory to physically determine the structure of a protein. Within their range of precision, these methods provide the definitive structure of a protein in atomic coordinates, against which other methods must be validated. There are also computational methods for predicting various aspects of structure, given the primary amino acid sequence. All of these methods have serious limitations. In the next We will also introduce another computational method called homology modeling, whose accuracy we will attempt to improve using our framework.

5.4.1 Laboratory Methods

One of the primary methods for physically determining the structure of a protein is X-ray crystallography (Bax, 1989). For crystallographic analysis, a protein must first be expressed, purified, and crystallized. X-rays are directed into the crystal, and diffraction patterns are captured on the other side with photographic plates. Within the crystal, the individual protein molecules line up in a three-dimensional lattice with complex symmetries. The repeating pattern of atoms in different molecules at roughly the same distance as the wavelength of the X-rays causes the beam to diffract, producing alternating dark and light patterns on the plates. X-ray crystallography has two major drawbacks. First, it can be quite challenging to properly prepare a high quality protein sample, sometimes taking years of lab work to devise a scheme for producing pure crystals. Also, the interpretation of the diffraction data is under-constrained, which is a significant problem because there are a huge number of possible three-dimensional conformations and hence potential models to consider. Highly compute-intensive programs have been written to facilitate crystallographic data analysis (Glasgow *et al.*, 1993), but it still can take months before a structure is confidently identified.

Nuclear magnetic resonance (NMR; Richards, 1992), another physical method for determining protein structures, also suffers from difficulties in obtaining material samples. However, the purification step is less demanding because only a concentrated solution of the protein is needed, not crystals. The solution is placed in a strong magnetic field, which uniformly orients the spins of the nuclei of all the hydrogen atoms in the protein molecule. Then a perpendicular electric field is applied, and its frequency is varied over a range. NMR experiments collect data in the form of correlations of responses between hydrogens on different side-chains. While these atoms may be separated by many covalent bonds, they may still be close enough in space to affect each other's resonance response. The degree of correlation is related to the distance between the atoms, so the data can be used to produce a rough pairwise distance matrix between residues. This matrix can be used to evaluate potential models of the three-dimensional conformation, although the analysis, as in crystallography, is generally under-constrained (Hayes-Roth *et al.*, 1986). The main limitation of NMR is that large proteins have so many electron pairs that the signals overlap and become uninterpretable. Currently NMR can only be used to determine the structure of proteins with less than about 150 residues.

5.4.2 Computational Simulation of Folding

Given the difficulties in sample preparation and data analysis, physical determination of structure can be slow if not impossible for many proteins. As an alternative, computational methods attempt to predict the structure of a protein from its amino acid sequence, rather than measure it. The most common prediction method is called molecular simulation (McCammon and Harvey, 1987). Molecular simulation is carried out by assigning three-dimensional coordinates to each atom in a protein molecule based on a random initial configuration. Then the position and velocity of each atom is incrementally adjusted using equations of electrostatics and Newtonian mechanics to compute the forces exerted by all the other atoms. Ideally, a simulation would be run until the structure converged on a stable configuration of minimal energy, perhaps using simulated-annealing techniques to avoid local minima.

One problem with molecular simulation is that the forces on each atom must be modeled from first principles of physics. Physicists do not yet have a compete theory of these forces, and the current equations must often be numerically approximated for computational purposes anyway. An additional complication is that solvent molecules must be incorporated into the simulation, for example to model the entropic forces behind the hydrophobic effect. These extra atoms greatly increase the complexity of the simulation. The most serious limitation of molecular simulation, however, is that proteins fold in continuous real-time, so the discrete time steps of the simulation naturally introduce some error. The only way to minimize this error is to decrease the interval of the time steps, usually to the range of picoseconds. But solving the equations of motion in each iteration is computationally intensive, and even state-of-the-art supercomputers running highly optimized parallel programs can take days to simulate a total of as little as one nanosecond. Since protein folding can take up to several seconds of real time, accurate determination of structures with globally minimum energy by molecular simulation *de novo* is intractable (Richards, 1992).

5.4.3 Secondary Structure Prediction

Another computational method for obtaining structural information about proteins from their amino acid sequences is secondary structure prediction. The goal of this approach is to predict the secondary structure–usually alpha-helix, beta-sheet, or random coil at each residue in a sequence. Many different algorithms have been invented for predicting secondary structure. One of the earliest approaches combined simple statistics for residue preferences with rules for nucleating, propagating, and terminating contiguous stretches of residues in a particular conformation (Chou and Fasman, 1974). Since then, the statistics have been replaced by elaborate neural networks (Qian and Sejnowski, 1988), and the rules have grown into large knowledge bases of the roles specific amino acids can play in determining secondary structure (Lim, 1974).

The accuracy of secondary structure prediction algorithms can be tested on individual sites in the amino acid sequences of proteins whose structures have been determined. One of the highest accuracies achieved to date is 64.4% (Stolorz *et al.*, 1992), although improvements are being made by choosing different definitions of secondary structure classes (Zhang and Waltz, 1993). Long range interactions may be causing intrinsic inaccuracy on this task, since these methods base their predictions only on sequentially adjacent residues, possibly missing the influence of residues that are globally adjacent in the 3D structure.

However, even highly accurate secondary structure prediction would not necessarily solve the protein tertiary structure prediction problem. It is unclear how predictions of local structure (at individual sites) can be assembled into a global three-dimensional structure for the whole protein. Bohr et al. (1990) present an interesting variation on secondary structure prediction that has been adapted to generate global information with some success. This method also uses a sliding window to predict local conformations, although secondary-structure classes are re-defined in terms of contact patterns, rather than the usual alpha/beta/coil distinction. These predictions are used to partially fill out a matrix of distances between each pairs of residues, which is then completed using a neural network. The final matrix is processed by a constraint-satisfaction procedure to produce a structural model of the protein.

5.5 Homology Modeling

Another computational approach to predicting the tertiary structure of a protein based on its amino acid sequence is *homology modeling.* Homology modeling refers to the process of aligning the sequence of a protein whose structure is unknown to the sequence of a protein whose structure is known. Provided their sequences are sufficiently similar, an approximate structure can be inferred by analogy (Blundell *et al.*, 1987). Homology modeling can be adapted as a general method for structure prediction by aligning a sequence with all the proteins in a protein database to identify the closest match. Databases like the Brookhaven Protein Databank (PDB; Bernstein *et al.*, 1977) now contain entries for hundreds of proteins whose atomic coordinates were empirically determined by physical methods. Homology modeling takes advantage of these valuable databases to learn the relationship between sequence and structure, in contrast to molecular simulation, which computes a structure from first principles.

Homology modeling is founded on the principle that *proteins with similar sequences have similar structures.* Sequence similarity is typically measured by the percent of amino acids that are identical between two sequences. It was recognized early on that proteins with a high percentage of amino acids in common appeared strikingly similar by visual inspection (Perutz *et al.*, 1965). Structural similarity can be quantified in terms of RMSD, the root mean square distance between corresponding atoms when two protein structures are superimposed (McLachlan, 1972). By comparing a wide range of proteins, Chothia and Lesk (1986) empirically found the relationship between sequence and structural similarity to fit the equation $RMSD = 0.4e^{1.87(1-H)}$, where H is the percentage of identical residues, implying that proteins with higher sequence similarity have lower RMSD.

In this thesis, we will be concerned with a discrete version of homology modeling in which protein similarity is defined qualitatively by fold class. All proteins in the same fold class have a low RMSD when compared with each other; Sander and Schneider (1991) found that proteins in the same fold class typically have an RMSD of less than around 3.0 angstroms. So the goal of protein tertiary structure prediction can be reduced to simply *identifying the fold class* of a protein, which has many fewer degrees of freedom than a complete set of atomic coordinates (Jones and Thornton, 1993). After the fold class of a protein has been identified, its sequence may be aligned to another member of the same class to derive an approximate structure, and the predicted atomic coordinates may subsequently be refined by molecular simulation (Nell *et al.*, 1992).

One potential criticism of the homology modeling approach to protein structure prediction is that it cannot be used to predict the structure of a truly novel protein. If there are no previous examples in the database of the fold class for a given protein, then no significant match for its sequence will be found. However, Chothia (1992) has estimated from a statistical analysis of current databases that there may only be around 1000 folds used in all biological systems. This limit is also supported by a theoretical study of the number of ways in which secondary structures like alpha-helices and beta-sheets can possibly be combined in a typical-sized protein (Finkelstein and Ptitsyn, 1987). Thus databases like the PDB may eventually grow from the approximately 100 distinct folds currently represented into a complete library for homology modeling.

Homology modeling has been the most successful computational approach to protein tertiary structure prediction to date. However, even homology modeling has some limitations. In the rest of this chapter, we discuss the central issue of homology modeling: how to generate and evaluate sequence alignments. Then we will introduce the main limitation of homology modeling, and conclude with a survey of previous attempts to address this problem.

5.6 Measuring Sequence Similarity

The most common method for measuring the similarity between two amino acid sequences, assuming their lengths are the same, is to line up their residues and count the number of matches. Averaged over the length of the two sequences, the percent amino acid identity is a simple but standard measure of sequence similarity. This measure is relevant to protein structure prediction because it reflects the processes of molecular evolution (Nei, 1987). When a gene evolves along two separate paths (for example, in two different species diverging from a common ancestor), the amino acid sequences undergo mutations that differentiate them. The most common mutation is the replacement of one amino acid for another, called *substitution*, which results in a mismatch between the two sequences. As substitutions accumulate over time, the structures slowly diverge (staying in the same fold class) in correlation with decreasing percent amino acid identity.

Percent amino acid identity is the basis for programs like BLAST (Altschul *et al.*, 1990), which retrieve protein database entries that contain sub-sequences with high similarity to segments of the query. These programs require a method for testing the significance of similarity scores to determine when they signal a structural relationship. Most pairs of sequences will have a few spurious matches that contribute somewhat to the percent amino acid identity. For example, two randomly generated sequences of the 20 amino acids can be expected match about 5% of the time. Unrelated sequences in the protein databases actually have slightly higher similarities, partly due to non-uniform distributions of the amino acids (Doolittle, 1981). Thus, a significance test is needed to determine whether a given score is high enough to be confident that they are evolutionarily divergent and hence structurally similar.

An early method for theoretically estimating the significance of a percent amino acid identity score was based on stochastic modeling (McLachlan, 1971). The significance of a match score between two sequences is taken to be the cumulative probability that random sequences with the same amino acid composition would achieve that score or higher. This probability can be calculated by using a polynomial representation to encode the range of possible match scores. The binomial distribution is used to compute the coefficients, which correspond to the probabilities of various scores. The prior probabilities of two particular residues being matched are taken from the relative proportions of the residues observed in the original sequences, making the calculation of significance sensitive to the particular distribution of amino acids in the proteins being compared.

While percent amino acid identity is a useful measure of sequence similarity, it does not account for another important class of mutations: insertions and deletions. During the evolutionary divergence of two protein sequences, random segments of amino acids can be inserted or deleted at various points, accounting for the variation in length among proteins in the same fold class. Insertions and deletions are typically one to ten residues long (Doolittle, 1981), and are most often found in the loop regions at the surface of a protein (Blundell *et al.*, 1987). Insertions or deletions in the tightly packed core of a protein are more disruptive and can completely destabilize a structure, whereas the insertion or deletion of amino acids at the surface are more easily accommodated.

A more sophisticated sequence comparison algorithm that accounts for insertions and deletions must be used to compute a similarity score between sequences with different lengths. A sequence-alignment algorithm can adjust two sequences to be the same length by inserting gaps at various points in one or the other (or both; Sankoff and Kruskal, 1983). The gaps make up for insertions and deletions, and can re-align local segments that have a high percentage of amino acid matches (few substitutions), re-constructing both kinds of mutational events. To prevent gaps from being inserted gratuitously to gain arbitrary matches, a penalty for each gap is subtracted from the percent amino acid identity. Since insertions and deletions are rarer mutations that substitutions, the cost of introducing a gap is typically higher. Also, since shorter gaps are more frequent the longer ones, the penalty for a gap is often taken to be a linear function of its length. Fitch and Smith (1983) analyzed of the effect of various gap penalties on alignments of sequences whose structures were known, observing that, while different gap-weight parameters produced slightly different alignments, the changes were minor over a wide range of values. They concluded that, as long as the gap-open penalty is several times greater than the cost of a substitution and the gap-extension penalty is several times smaller, the alignments that are generated reasonably reflect the correspondence between amino acids in the structures.

The challenge of sequence alignment is to find a configuration of gaps that makes an optimal tradeoff with local matching to produce a globally maximum score. One of the first algorithms for efficiently constructing optimal alignments used dynamic-programming to achieve time- and space-complexities of $O(n^2)$ (i.e. proportional to the product of the lengths of the two sequences; Needleman and Wunsch, 1970). This algorithm was later extended to incorporate linear gap weights (Smith and Waterman, 1981; Gotoh, 1982). Another improvement in the sequence alignment algorithm was to reduce the memory requirements to linear space-complexity, which can be important when aligning long pairs of sequences (Hirschberg, 1975; Myers and Miller, 1988). Parallel implementations have also been studied, in order to take advantage of massively-parallel supercomputer architectures (Jones, 1992).

The significance of an alignment score is much harder to determine theoretically than the significance of the percent amino acid identity between two protein sequences without gaps (Fitch, 1970; Steele, 1982). A more practical method for estimating the significance of an alignment score, called "shuffling," is based on simulation (Doolittle, 1981). Given the alignment score between two sequences, we want to know how likely it is that they could match that well without being evolutionarily related. To answer this question, we could shuffle the sequences (i.e. independently randomize their orders), purposely destroying any evolutionary correspondences while retaining the same amino acid compositions. A sampling of the similarity scores between shuffled sequences forms a distribution that is approximately normal and can be used to compute a *z-score* for the original alignment, in the standard statistical sense (Lipman and Pearson, 1985).

One disadvantage of shuffling is that it relies on various assumptions about the independence of mutations, which might not be an accurate model of molecular evolution. A different method for empirically testing the significance of an alignment score is to compare it to a distribution of scores from alignments of the proteins of interest with other proteins in the database that have different structures. If the two proteins are indeed unrelated, then their alignment score should follow the scores from comparisons with other such proteins. Although this distribution is not necessarily normal, alignment scores more than three or so standard deviations above the mean are generally considered to be significant (Gribskov *et al.*, 1988). While this method for estimating the significance of an alignment score makes fewer assumptions about evolutionary processes, one drawback is that it is sensitive to the kinds of sequences represented in the database (such as primarily alpha proteins versus primarily beta proteins). Also, it is computationally demanding because it requires many additional alignments to be done in order to evaluate a single score.

5.7 The Hidden-Homology Problem

Like other computational methods for predicting the tertiary structure of a protein from its amino acid sequence, homology modeling has some limitations. The basic problem with homology modeling stems from the fact that, while proteins with similar sequences always have similar structures, the reverse does not always hold true. Recently it has been observed that some proteins with similar structures do not appear to have significant sequence similarity (Jones and Thornton, 1993). We call this phenomenon the "hidden-homology problem." It is unclear whether these proteins originated from a common ancestor and subsequently diverged so greatly that all evidence of evolutionary relationship was erased, or whether they converged independently to the same stable structure, re-used for different functions (Doolittle, 1981; Finkelstein and Ptitsyn, 1987).

A prime example of the hidden-homology problem is the comparison between mandelate racemase and muconate lactonizing enzyme, which have only 26% percent identity in their amino acid sequences. This similarity score was not high enough to suggest any evolutionary relationship, and hence there was no reason *a priori* to expect them to have similar structures. Yet when their structures were solved, they were both found to be members of the beta-barrel fold class, and their structures differed by an RMSD of only 1.3 angstroms (Neidhart *et al.*, 1990). In fact, there are at least 14 mutually distinct beta-barrels known, including the prototypical triose-phosphate isomerase (TIM-barrel), as well as more diverse proteins such as taka alpha amylase and tryptophan synthase (Chothia, 1988).

The observation that proteins with similar structures do not necessarily have similar sequences is a major drawback for homology modeling. In particular, the hidden-homology problem causes false-negative predictions. If all the database examples for the fold class to which a query protein actually belongs happen to have low similarity to the query sequence, then no match will be detected, and homology modeling will fail to identify the fold class. As a result, one or a few example proteins may not be a sufficient representation for a fold class, and the possibility of building a complete library of folds becomes unlikely. Thus the hidden-homology problem is causing under-utilization of our valuable databases. Almost certainly there are sequences that belong to known fold classes, but whose structures remain unknown because they do not appear similar to the available examples. Although the extent of this problem has not yet been determined, there are a vast number of proteins whose structures are unknown that have no more than 20-30% amino acid identity with any protein whose structure is known. Doolittle (1986) has called this range of sequence similarity scores the "twilight zone" because they can be used neither to confirm nor deny structural relationships.

5.8 Previous Attempts to Improve Homology Modeling

Several different approaches have been taken to address the hidden-homology problem. A common technique is to use the frequency of amino acid substitutions as partial match scores to improve alignments. Other methods increase the sensitivity of alignments by deriving assistance from extra sources of information. The known structures of database proteins can be exploited to evaluate the fit of an aligned sequence. And multiple examples of a fold class can be generalized into profiles or templates for improved pattern recognition. In the next chapter, we will introduce another approach that takes advantage of knowledge about the properties of amino acids.

5.8.1 Amino Acid Substitution Matrices

Dayhoff et al. (1972) were among the first researchers to notice certain biases in the frequencies with which various amino acid substitutions occur in alignments of evolutionarily related proteins. The distribution of replacements is not uniformly random, and cannot entirely be explained by the relative proportions of the residues found in proteins in general. The replacing amino acid strongly depends on the residue being replaced, perhaps reflecting a need to conserve certain physical or chemical properties. So Dayhoff compiled pairwise tables of statistics on the probabilities of substitutions observed in multiple alignments. The score between each pair of amino acids can be seen as an kind of "empirical" similarity measure.

Substitution matrices can easily be incorporated into most sequence alignment algorithms, using the frequency of substitution between each pair of residues as a partial match score

(Sankoff and Kruskal, 1983). Instead of penalizing all mismatches equally, this approach partially rewards mismatches between residues that are observed to exchange frequently. The use of such partial match scores should help distinguish distantly related proteins (with similar structures but low sequence similarity) from truly unrelated proteins. Mismatches between unrelated proteins should not be biased, but should follow a distribution expected from the relative proportions of the residues. Such alignments would not benefit as much from the partial match scores, having a balance of likely and unlikely substitutions.

There are several problems with amino acid substitution matrices that inhibit their utility for homology modeling. First, the substitution frequencies are highly sensitive to the particular sample of proteins in the multiple alignments used to generate the statistics. Current databases under-represent certain kinds of proteins, such as membrane proteins, which are difficult to crystallize for structure determination. Second, substitution frequencies depend on the evolutionary distances between aligned proteins (Dayhoff *et al.*, 1978). Third, the scalar value of similarity between each pair of residues is incapable of expressing more complex relationships among the amino acids. The similarity between any two amino acids at a given site depends on the structural role being played, and different physical and chemical properties are emphasized in different contexts (Overington *et al.*, 1992). Finally, the statistics in a substitution matrix reflect an overall bias toward evolutionarily-related proteins, since these are the ones that are typically used to construct multiple alignments. Thus the substitution frequencies might be more representative of the forces of molecular evolution than of the similarity in structural roles that the amino acids can play. Henikoff and Henikoff (1993) present a method for evaluating the performance of various substitution matrices for protein structure prediction.

5.8.2 Evaluating Fold Potentials

Whereas the use of amino acid substitution matrices is an attempt to facilitate homology modeling by improving the alignment score between sequences, some researchers have begun to exploit the structural information associated with each entry in a protein database as well. One approach is to fold the query sequence into the shape of the candidate structure, and then evaluate the "quality" of the fit. This approach is often used to solve the *inverse folding problem*, in which possible sequences are generated for a given structure (Bowie and Eisenberg, 1993).

For fold-class prediction, the evaluation of fit provides an independent indexing mechanism for identifying structures in a database that a query sequence could potentially match.

One of the difficult aspects of evaluating fold potential is the initial step of folding the sequence into the candidate structure, also called threading (Lathrop and Smith, 1996). This structural alignment suffers the same complications caused by insertions and deletions as in sequence alignment (Taylor and Orengo, 1989). Bowie et al. (1991) have shown how structural information at each site can be encoded qualitatively in the sequence of a database protein by supplementing the amino acid identities with an indication of "environment class." This method allows standard alignment algorithms to accomplish threading.

Once threading is completed, the fit of the sequence can be evaluated by a variety of methods that can take global properties into account. For example, Ponder and Richards (Ponder and Richards, 1987) used packing density as a criterion, since proteins pack uniformly very tightly, without any cavities (or steric overlaps). Another useful criterion is based on the solvation preferences of the residues, which should be reflected in the partitioning of residues between the surface and core of the protein (Miller *et al.*, 1987). Yet another criterion is the propensity for each pair of residues to be near each other, which can be transformed into an energy function for evaluating the observed spatial distribution of residues (Sippl and Weitckus, 1992).

5.8.3 Profiles and Templates

Another approach to improving the recognition of fold class identity from an amino acid sequence is to incorporate information from multiple examples. Gribskov et al. (1988) describe how to create a profile for a fold, consisting of individual distributions of residues observed at each site, from a multiple alignment of various members of the fold class. The idea behind profiling is that, by looking at the distribution of amino acids at each site across multiple examples, one can infer general constraints on amino acid substitutions that are more directly relevant to the fold as a whole than to any specific example sequence. There are two main steps to profiling. First a multiple alignment is constructed to provide a consistent definition of sites among all the proteins. Then the distribution of residues (and gaps) at each site is collected and smoothed using a general substitution matrix to construct a position-specific profile of residue propensities.

To compare a new sequence to the profile, local match scores are generated by looking up the propensity of the observed residue for each specific site in the profile. So, like amino acid substitution matrices, profiling enhances sequence alignment by giving partial match scores. However, instead of using a single distribution that must universally represent substitution patterns for all sites, profiling allows each site to have its own specialized distribution. The distribution at each site reflects the unique constraints placed on the amino acids at that site based on the structural role they play in the local environment. Profiling is most successful when there are many diverse examples of a fold because evolutionary biases are diluted out of the residue distributions, leaving only the structural constraints at each site.

Like profiles, templates are patterns generalized from multiple sequences in a fold class. However, template extraction methods attempt to focus pattern-recognition on only the most highly conserved regions of a protein, identifying diagnostic "fingerprints" (Taylor, 1986). For example, the PIMA method of Smith and Smith (1990) uses a property-based hierarchy of amino acids to generalize the residues found at each site. For example, the parent of the leaf node for aspartate is {aspartate,glutamate}, whose parent is the set of nine charged residues, whose parent is the root node: 'any'. The larger the set of amino acids found at a site, the higher the node in the tree required to minimally cover them. When comparing a new sequence with a template, matches are given scores inversely proportional to the height of the node in the hierarchy. Thus the symbols act as discrete versions of the distributions in profiles, but are extra-restrictive to filter out less informative sites.

ARIADNE is another template-based program for generalizing multiple examples of a fold (Lathrop *et al.*, 1987). ARIADNE uses a hierarchical pattern-language to represent folds at multiple levels of description. For example, the lowest level consists of the amino acid identities, but subsequences may be annotated with secondary structure information (from solved structures), and contacts among these secondary structures (super-secondary structures, motifs) may be indicated at an even higher level. ARIADNE uses a sophisticated pattern-matching algorithm (rather than sequence alignment) in order to utilize the expanded range of sources of knowledge about protein structure made available by the hierarchical representation. The actual generation of fold patterns from examples is carried out by ARIEL, a massively-parallel program that implements inductive heuristics to extract commonalities among hierarchical descriptions (Lathrop *et al.*, 1990).

Chapter 6

Experiments

In this chapter, we will demonstrate the framework by applying it to improve homology modeling, particularly by addressing the hidden-homology problem. As background knowledge in this domain, we will use the chemical and physical properties of amino acids, which are relevant to protein structure. We will show how amino acid properties can be used to change the representation of protein sequences, which is a crucial step in applying the framework. We will then describe several experiments that use this knowledge to search for alternative representations of amino acid sequences that increase the ability of homology modeling to detect structural similarities among proteins.

6.1 Applying the Framework

In accordance with our framework, we will need to specify several resources in this domain. First, we must select an induction algorithm whose predictive accuracy is to be improved. Then we must identify some relevant domain knowledge. Finally, we must adapt this knowledge so that it can be used to suggest alternative representations.

6.1.1 Treating Homology Modeling as a Learning Algorithm

Our framework presumes that there exists an induction algorithm that will remain fixed during the course of the experiments. Homology modeling (with a protein database) is a domain-specific method for predicting protein structures which we would like to improve using our framework. As shown in Table 6.1, homology modeling can be thought of as a nearest-neighbor

Table 6.1: Analogy between inductive learning and homology modeling.

inductive learning	*homology modeling*
training examples	protein database
description language	amino acid sequences
classifications	fold classes
inductive bias	nearest-neighbor
similarity metric	alignment score

learning algorithm (Aha *et al.*, 1991). The proteins in the database are the examples, described initially as amino acid sequences. They are labeled with their known fold-classifications. Given a new sequence, it is compared to all the database entries by doing sequence alignments. If a match with significantly high alignment score is found, then a prediction is made by inferring that it has the same fold class (approximate structure) as the matched entry. Thus all the previous examples are saved without generalization, and new sequences are classified the same as the closest previous instance, using sequence alignment scores as a specialized similarity metric.

6.1.2 Knowledge About Amino Acid Properties

Our framework for change-of-representation relies fundamentally on the availability of domain knowledge. In this domain, biophysicists have extra background knowledge that is believed to be relevant to protein structure. This knowledge consists of the chemical and physical properties of the 20 amino acids, based on their unique side-chains. Physical properties refer to the structure of the side-chains, such as being branched, bulky, flexible, short, etc. Chemical properties are determined by atomic constituents and their covalent bonding, such as being charged, hydrophobic, aromatic, or reactive. It is these properties that determine the role each amino acid plays in a protein structure, and hence we want to take advantage of this knowledge to increase the quality of sequence alignments.

The importance of amino acid properties was first noticed by Dayhoff and co-workers (Dayhoff *et al.*, 1972). They observed non-uniform patterns of substitutions of amino acids in multiple alignments (among proteins with similar structures). In comparison to the expected rates of substitution based on the natural distribution of amino acids, certain replacements were found to be highly preferred, while others occur at much lower frequencies. Dayhoff showed that

Table 6.2: Exchange groups and their interpretation as properties by Dayhoff et al. (1972). The groups are presented as sets of amino acids in their 3-letter abbreviations.

amino acid group	*property name*	*description*
met ile leu val	hydrophobic	large and non-polar
phe trp tyr	aromatic	ring with double-bonds
his arg lys	positive	charged by proton
ala asp glu gly asn pro gln ser thr	small	side-chain volume
cys	disulfide	can form cross-links

replacements tended to fall within disjoint subsets of amino acids, which she called *exchange groups* (see Table 6.2). The replacement of one amino acid often came from the same exchange group, which defines a *conservative* substitution. Consequently, the set of amino acids found at a given often resembled one of these five groups.

The existence of exchange groups suggests that conserved sites involve a structural constraint that restricts some replacements and allows others. By looking for general characteristics which are common to the members of a particular group, but not found in the other amino acids, Dayhoff proposed labels for the exchange groups (also listed in Table 6.2). HYDROPHOBIC includes the large, non-polar residues; AROMATIC means the side-chain contains a conjugated ring; POSITIVE refers to residues that are electrically charged by protonation. Cysteine is in its own group because it is the only residue that can cross-link a structure by forming DISULFIDE bridges. The remaining residues share the property of being relatively SMALL, in terms of side-chain volume. Based on this interpretation of the exchange groups, Dayhoff inferred that the forces that determine the structure of a protein are primarily a function of these five chemical and physical properties.

Taylor (1986) generated an expanded list of properties by taking certain subsets and supersets of Dayhoff's exchange groups, as well as arbitrary intersections and unions of groups. Some examples of Taylor's properties are TINY, which is a particular subset of the SMALL residues, and NEGATIVE, which equivalent to CHARGED-AND-NOT-POSITIVE. These properties are not derived from observed substitution patterns, as were Dayhoff's properties, but they are considered to be *plausible* properties that might be playing a special role at certain sites in proteins.

Richardson and Richardson (1989) present a detailed survey of amino acids, cataloging their unique properties along with experimental evidence for the roles they play in protein structures. They offer explanations for substitution patterns in terms of specific features of side-chains. For example, they discuss the ability of asparagine to mimic the backbone, which helps cap-off helices, and the ability of residues like valine with a branch at the beta-carbon to support structures that pack against it, which is especially useful in beta-sheets.

Some research has focused on independently ranking the 20 amino acids along various scales. For example, many different definitions for hydrophobicity have been proposed, based on criteria ranging from theoretical calculations of polarity to empirical measurements of partitioning between various solvents (Cornette *et al.*, 1987). Kidera et al. (1985) analyzed 188 amino acid property scales gathered from the literature and found that they clustered into nine general groups. A factor analysis of the groups revealed that most known scales are a combination of four fundamental properties: bulkiness, hydrophobicity, helix-preference, and sheet-preference.

6.1.3 Re-representing Amino Acid Sequences

The preference for substitutions between amino acids with similar properties illustrates the redundancy that lies at the heart of the hidden-homology problem. Because the structural role being played at a given site can often be satisfied by multiple residues, the identity of the residue is over-specific. Instead, it would be more representative to express the main property of that residue, since it is the chemical and physical properties that actually determine how an amino acid participates in the local structure (surrounding protein environment). This suggests that a better representation of a protein than its sequence of amino acid identities would be its sequence of properties, which more faithfully reflects the local structural constraints holding the protein together.

Replacing the identities of amino acids with their physical and chemical properties should facilitate homology modeling because it provides more information for local matching during sequence alignment. If the properties of amino acids are explicitly encoded in the sequence, then the pair of amino acids at a site can be matched if they have the same properties. Hence even non-identical amino acids may be counted as a match if they can fulfill the same role in the local structure. The enhanced matching could help alleviate the hidden-homology problem because proteins with similar global (3D) structures must be composed of sequences of sites with similar

```
Met - Ile - Arg - Cys - Gly - Asn - Leu - Arg - Phe
 |     |     |     |     |     |     |     |     |
 V     V     V     V     V     V     V     V     V
HYD - HYD - POS - DIS - SML - SML - HYD - POS - ARO
```

Figure 6.1: Example of applying the Dayhoff transformation. The top sequence is an example segment of protein sequence in its original representation, amino acid identities, and the bottom sequence is the re-representation of the example, with elements mapped into a new alphabet. The amino acid identity at each site is replaced by a symbol for its property name, as interpreted by Dayhoff (the mapping is given in Table 6.2). In this figure, we use the following 3-letter symbols for the properties: HYD=hydrophobic, ARO=aromatic, POS=positive, SML=small, DIS=disulfide.

local structures. Thus this re-representation should improve the ability to recognize proteins in the same fold class by capturing more subtle patterns based solely on structural constraints, rather than evolutionary artifacts (i.e. amino acids identities not yet mutated).

Formally, we can capture such changes-of-representation as *sequence transformation functions*. A sequence transformation S is a function that maps strings over one alphabet A into strings over another alphabet B: $S : A^* \mapsto B^*$ In the context of this thesis, we will be interested in length-preserving transformations ($|S(a)| = |a|$ where $a \in A^*$), so there will be the same number of symbols in a re-represented protein sequence as there are amino acids (or sites) in the original sequence.

The simplest sequence transformations involve re-naming of symbols. For example, consider mapping amino acid identities to the property names suggested by Dayhoff (see Table 6.2). The initial alphabet consists of 20 symbols (the amino acids), and the final alphabet consists of 5 symbols (HYDROPHOBIC, AROMATIC, POSITIVE, SMALL, DISULFIDE). The definitions given by Dayhoff show how to map each amino acid symbol to a specific property symbol. This re-naming is uniformly applied to each site in a protein sequence to generate the re-represented sequence. We call this the *Dayhoff transformation* (an example is shown in Figure 6.1). Other sequence transformation functions can be more complicated, mapping into extended alphabets that encode various combinations of properties, and the mapping itself can depend on things such as the position in the sequence or properties of other residues in the sequence, not just on the identity of the amino acid itself.

6.1.4 Uncertainty in the Domain Knowledge

The problem with this domain knowledge is that biophysicists are unsure which amino acid properties are most important for protein structure prediction. Dayhoff et al. (1972) noted some anomalies in the substitution patterns that caused difficulties in identifying disjoint exchange groups. Some amino acids are replaced with appreciable frequency by members of two or more groups. For example, histidine substitutes with AROMATIC as well as POSITIVE residues. Similarly, residues within a group do not always substitute well for other members of that group. For example, glycine is often replaced by alanine, but not by proline; yet all three of these residues were placed in the SMALL group.

Taylor's (1986) plausible descriptions of amino acid properties also raise some interesting questions about the best way to define them. For example, he leaves methionine out of the HYDROPHOBIC group because it contains a partially-polar sulfer atom. Taylor also includes histidine with the AROMATIC residues because it contains a conjugated ring. But histidine is different from phenylalanine, tyrosine, and tryptophan in that its ring contains non-carbon atoms (i.e. it is heterocyclic). Taylor also uses a series of subsets to express various degrees of certain properties. For the POSITIVE residues are contained in the CHARGED residues, which are contained in the POLAR residues. Biophysicists do not have strong enough knowledge about the forces determining protein structures to decide which definitions are best among these plausible alternatives.

These problems stem from fact that amino acids have multiple properties that generate overlapping dimensions of similarity. For example, threonine has a shape like valine (i.e. branched), and is hydroxylated like serine. But valine and serine are not similar to each other in either of these ways. Notions of amino acid similarity tend to compress these distinctions into simple relationships (such as the pairwise scalar values in a substitution tables; Dayhoff *et al.*, 1972), but each of the individual properties might play a role in some situation. The relevant property at a given site is probably *context-dependent*, since substitution patterns are known to be highly influenced by the surrounding tertiary environment (Overington *et al.*, 1992). So, despite studies such as by Kidera et al. (1985), it is difficult to say which properties are most important because different properties may be important in different contexts.

Due to these uncertainties, a single re-representation of amino acid sequences in terms of properties cannot easily be chosen. Our approach, based on our framework, is to let the uncertainties about amino acid properties generate a space of alternative representations. For example, if there are competing definitions of a property, then we can try using each one independently in a sequence transformation function and pick the one that is best for homology modeling. Similarly, if we do not know the best context for a given property, we can try restricting the transformation to only a subset of sites to focus its effects. This approach flexibly combines the use of domain knowledge with optimization where the knowledge is weak.

6.2 Experimental Setup

6.2.1 Overview of Experiments

In the rest of this chapter, we describe two experiments that test these ideas for change-of-representation in the protein structure prediction domain. In each case, we give a domain theory in first-order logic encoding the biophysical knowledge supporting the experiment. From these constraints, we derive an efficient search procedure, the implementation of which we describe in LISP. We also supply some related analyses of the meaning of the results of each experiment for protein science.

6.2.1.1 Preview of Experiment 1

- **Purpose:** Re-naming amino acid identities with their properties may facilitate homology modeling. Dayhoff's exchange groups provide one particular description of properties, but these may be slightly incorrect. So we are willing to consider slight variants of this partition to find the best set of definitions of amino acid properties for homology modeling.

- **Search:** The idea of variant partitions leads to the definition of an operator that swaps amino acids randomly one at a time to new classes in the partition. Independence assumptions translate into a Monte Carlo form of hill-climbing through the space of partitions of amino acids using this operator.

- **Conclusions:** We will see that the Dayhoff partition turns out to be worse for homology modeling than amino acid identities. Furthermore, the search process never *on average* finds alternative partitions that are better than identities. However, the Dayhoff partition itself is significantly improved, showing that knowledge of variants, coupled with the method of empirically evaluating and comparing representations, can improve some representation (i.e. the Dayhoff partition). This establishes the basis for the approach in Experiment 2.

- **Analysis:** Sets of improved partitions can be generalized by looking for patterns of amino acids that are frequently found in the same class together. In comparison to the original Dayhoff classes, some properties appear to be stable (i.e. amino acid subsets are rarely perturbed during search), while other properties appear to need to be re-defined.

6.2.1.2 Preview of Experiment 2

- **Purpose:** Experiment 2 addresses the question of why Experiment 1 did not surpass the performance of homology modeling when using identities. Background knowledge suggests that different amino acid properties may be relevant in different contexts. Rather than forcing a set of properties to combine together in a single partition, individual properties are considered for re-naming identities in separate contexts, which are defined by the pattern of hydrophobicity among sequential neighbors of each site.

- **Search:** In this experiment, alternative representations are *sets* of partitions, each grouping amino acids together according to one of a set of pre-specified properties. Due to independence and incrementality assumptions among alternatives, the search implied by the background knowledge in this experiment consists of optimizing the contexts one at a time, picking the property that gives the most improvement.

- **Conclusions:** This experiment measures *average z-scores*, which is measure of the significance of fold-class predictions by homology modeling. Using the best context-dependent re-representation discovered during search, the average z-score for homology modeling is increased over identities, and even beyond the alignment method using the MDM78 substitution table (Dayhoff *et al.*, 1978).

- **Analysis:** We show that the new context-dependent representation improves the significance (z-score) of sequence alignments uniformly with respect to percent amino acid identity (both high and low), which is important for use in biology. However, we observe that, although the sensitivity of alignments is improved in a statistically significant way, relatively few new site matches are achieved in actual alignments, pointing out room for future improvements.

6.2.2 Protein Data

The 199 proteins used in the these experiments are listed in Table 6.3. The amino acid sequences were taken from the Brookhaven Protein Databank (PDB; Bernstein *et al.*, 1977). They are specified by their 4-character PDB names, along with chain identifiers and sub-sequence limits where necessary (defaults are: first chain in file, whole sequence). The sequences ranged in length from 39 (4gcr) to 478 (2taa). These proteins were selected for these experiments because they had been grouped into specific fold classes by visual inspection (Pascarella and Argos, 1992). The 199 sequences fall into 37 different folds, with between 2 and 29 members per fold. The fold classifications are also listed in Table 6.3.

6.2.3 Computation of Sequence Alignments

These experiments share a core module for doing homology modeling that is used in the evaluation component of our framework. To evaluate a representation requires a set of re-represented test sequences to be aligned with a set of training sequences. From these alignment scores, a variety of measures can be computed, such as predictive accuracy (e.g. when the closest training sequence to a test sequence has the same fold) or z-score (e.g. estimating significance of an alignment by normalizing with respect to scores between unrelated proteins).

So the central task is, given a set of re-represented sequences (including both training and testing sets), to compute the entire pairwise table of alignment scores. For our whole dataset, this requires on the order of 20,000 alignments ($n * (n-1)/2 = 19,701$ where $n = 199$). To accomplish this, we implemented a standard alignment algorithm on the CM5 massively-parallel supercomputer. The module was written in C using the CMMD message-passing library. Whole alignments were distributed as independent tasks to available processors, rather than parallelizing the alignment algorithm itself (Jones, 1992). On a 256-processor partition, doing all 20,000 pairwise alignments necessary to evaluate one representation takes approximately 30

Table 6.3: The protein data used in the experiments in this thesis. Sequences are specified by their 4-character PDB name (Bernstein et al., 1977), along with optional chain letters and sequence limits. The sequences were assigned to fold classes by Pascarella and Argos (1992).

fold class	example proteins
256B	2ccy(A), 256b(A)
AC-PROT	1cms(1-175), 1cms(176-323), 4ape(2-174), 4ape(175-323), 3app(1-174), 3app(175-323), 2apr(1-178), 2apr(179-325), 4pep(1-174), 4pep(175-326)
BARREL	2taa(A), 1wsy(A), 1tim(A), 1gox, 1ypi(A), 1fcb(A:100-511)
BINDING	2liv, 2lbp, 2gbp, 1abe
CARBONIC	1ca2, 2cab
CA-BIND	3cln, 5cpv, 3icb, 4tnc, 5tnc
GCR	2gcr(1-39), 2gcr(40-86), 2gcr(87-127), 2gcr(128-173), 4gcr(1-39), 4gcr(40-87), 4gcr(88-128), 4gcr(129-174)
CYTB	1fcb(A:1-99), 3b5c
CYTC	451c, 1ccr, 1cyc, 5cyt(R), 3c2c, 155c(1-122)
CYT3	2cy3, 2cdv
DFR	3dfr, 4dfr(A), 8dfr, 1dhf(A)
EGLIN	1cse(I), 2ci2(I)
FAD-NADH	1phh, 3grs(18-161), 3grs(186-294)
FCX	3fcx, 1ubq
GLOBIN	4hhb(A), 4hhb(B), 2mhb(A), 2mhb(B), 1fdh(G), 1mbd, 1mbs, 2lhb, 1eca, 2lh1, 1pmb, 1mba
HLA-A2	3hla(A:1-90), 3hla(A:91-182)
MHR	2hmz, 2mhr
IGB	2fb4(L:1-109), 2fb4(L:110-214), 2fb4(H:1-118), 2fb4(H:119-221), 2fbj(L:1-106), 2fbj(L:107-213), 2fbj(H:1-118), 2fbj(H:119-218), 1mcp(L:1-113), 1mcp(H:1-122), 1rei(A), 2rhe, 3hfl(L:1-105), 3hfl(H:1-116), 3hfl(H:117-213), 1cdh, 3hla(A:183-270), 3hla(B:1-99), 7fab(L:1-109), 7fab(L:104-214), 7fab(H:1-117), 7fab(H:118-220), 2f19(L:1-108), 2f19(L:109-215), 2f19(H:1-123), 2f19(H:124-220), 4fab(L:1-112), 3hfm(L:1-108), 1mcw(W)
IL	1i1b(3-51), 1i1b(50-107), 1i1b(108-153)
INHIBIT	1tgs(I), 3sgb(I), 2ovo, 1ovo(A)
LTN	3cna, 2ltn(A:1-108), 2ltn(A:109-181), 2ltn(B:1-47)
LZM	3lzm, 2lzt, 2lz2, 1lz1, 1alc
NBD	4mdh(A), 2ldb, 1ldm, 5ldh, 2ldx, 1llc, 8adh, 3gpd(R), 1gpd(G), 1gd1(R), 1fx1, 4fxn, 3adk, 8atc
PAP	9pap, 2act
PLIPASE	1pp2(R), 1bp2, 1p2p
KINASE	1pfk(A), 3pfk
PLASTO	2paz, 1plc, 1azu, 2aza(A), 7pcy, 7rxn
RDX	4rxn, 1rdg, 6rxn
REPRESSOR	1r69, 2cro, 1lmb(3)
RHD	1rhd(1-146), 1rhd(152-293)
SBT	1cse, 1sbt, 1tec(E), 2prk
S-PROT	1ton, 2pka, 2ptn, 2trm, 4cha(A), 3est, 1hne(E), 1sgt, 2sga, 3sgb(E), 2alp
TOX	2abx(A), 1nxb, 1ctx
VIRUS	4rhv(1), 4rhv(2), 4rhv(3), 4sbv(A), 2mev(1), 2mev(2), 2mev(3), 2tbv(A), 2stv, 2plv(1), 2plv(2), 2plv(3), 1r1a(1), 1r1a(2), 1r1a(3)
VIRUS-PROT	3hpv, 2rsp(A)
WGA	7wga(A:1-43), 7wga(A:44-86), 7wga(A:87-129), 7wga(A:130-171), 9wga(A:1-43), 9wga(A:44-86), 9wga(A:87-129), 9wga(A:130-171)
XIA	4xia(A), 1xya

seconds of runtime. These experiments involve between 100 and 1000 iterations, taking roughly between 1 and 10 hours of CPU time.

Because alignments of moderately long sequences (400-500 amino acids long) had to be done, and space for such alignments had to be available on each processor node, memory was a limited resource. Hence we used the linear-space global-alignment algorithm of Myers and Miller (1988).[1] We normalize the resulting alignment score by dividing by the minimum length of the two sequences, to convert the total number of matches minus gap costs into something more like a percentage of matching. Another approach would be to divide scores by the log of the product of the lengths, which was found to be appropriate for alignments of nucleotide sequences (Smith *et al.*, 1985).

We used the following gap-weight parameters, relative to match value of 1.0: gap-open-penalty=1.5, gap-extension-penalty=0.2. These parameters were selected by a simple optimization procedure[2] and held constant throughout the experiments. They are consistent with the observations of Fitch and Smith (1983), who recommend setting the gap-open-penalty to several times greater than the match value, and the gap-extension-penalty to several times smaller than the match value. Informal testing did not reveal much sensitivity of our results to the gap weights used.

6.3 Experiment 1

6.3.1 Purpose

Background knowledge in this domain suggests that expressing amino acid properties in protein sequences instead of identities might improve recognition of structural similarity among proteins because properties reflect the constraints of local structure. In this experiment, we examine the effect on homology modeling of transforming amino acids into their properties. Dayhoff proposed one set of properties, which happens to form a partition of the amino acids. However, because there was some uncertainty in this knowledge, it is possible that some variant partition

[1] Most alignment algorithms require quadratic space (i.e. proportional to the product of the lengths of the sequences being aligned) to store the matrices used for dynamic programming (Needleman and Wunsch, 1970). The Myers and Miller (1988) algorithm simulates the same computation with several parallel arrays.

[2] A grid of parameters was set up for a reasonable range of values for each of the two gap-weight parameter. The pair of values was chosen that maximized the average z-score over all pairs of sequences in the same fold class. The grid was shifted and focused several times to ensure that a maximum had been found.

contains better definitions of the properties. Thus we consider arbitrary partitions of amino acids as a parameterization of the space of potential changes-of-representation.

6.3.2 Background Knowledge

The background knowledge on which this experiment is based can be formally expressed in logic for use in a CORIL-like system. There are three sets of statements involved: 1) transforming amino acid sequences in sequences of amino acid properties should improve homology modeling, 2) the Dayhoff partition constitutes one plausible definition of properties which can be used to transform examples, and 3) perturbing the Dayhoff partition by swapping amino acids among classes might lead to even better representations.

First, we express the knowledge that proteins with similar 3D structures have similar sequences of local structures, which implies that their sequences of properties are probably similar, due to the constraints local structure puts on amino acid properties. In this formalization, we use the function term 'seq(A,B)' to mean the sequence of protein A where the element for each site expresses aspect B (e.g. identities, properties, local structures, contexts, etc.).

$$\begin{aligned}
&\forall_{X,Y}\ protein(X) \ \wedge \ protein(Y) \ \wedge \ fold(X) = fold(Y) \\
&\quad \rightarrow similar(struct3D(X), struct3D(Y)) \\
&\quad \rightarrow similar(seq(X, localStructs), seq(Y, localStructs)) \\
&\quad \rightarrow similar(seq(X, roles), seq(Y, roles)) \\
&\quad \rightarrow similar(seq(X, props), seq(Y, props)
\end{aligned}$$

Since homology modeling currently suffers from false negatives, it could be improved by any change that allows it to recognize more matches between pairs of proteins that have similar 3D structures. The existence of proteins with similar structures but dissimilar amino acid sequences (e.g. mandelate racemase and muconate lactonizing enzyme (Neidhart *et al.*, 1990)), along with the belief that the property sequences should be the same, leads to the inference that sequences of properties should be better for homology modeling than sequences of amino acid identities. Here we connect the theory to the 'order()' predicate, which stands for the partial order among alternative representations that the expert believes should improve learning.

$$\exists_{X,Y}\ fold(X) = fold(Y)\ \wedge\ \neg similar(seq(X, idents), seq(Y, idents))$$

$$\forall_{A,B}(exists_{X,Y}\ similar(seq(X, A), seq(Y, A))\ \wedge\ \neg similar(seq(X, B), seq(Y, B)))$$

$$\rightarrow betterForHomologyModeling(A, B)$$

$$\rightarrow order(A, B)$$

One representative set of properties is given by Dayhoff (see Table 6.2). We encode the primary property of each amino acid in logical assertions. Given that these properties form a partition of the amino acids, we can now specify what in meant by a property sequence (seq(A,props)) by describing how examples are transformed: each element in the sequence is replace with the name of its class in the partition.

$$prop(ala, small),\ prop(cys, disulfide),\ prop(asp, small),\ prop(phe, aromatic)...$$

$$prop(A, B) \rightarrow class(A, dayhoff) = B$$

$$similar(seq(X, props), seq(Y, props))$$

$$\rightarrow similar(seq(X, dayhoff), seq(Y, dayhoff))$$

$$\rightarrow similar(transform(seq(X, idents), dayhoff), transform(seq(Y, idents), dayhoff))$$

$$transform(Seq1, Part) = Seq2\ \wedge\ (1 < N < length(Seq1))$$

$$\rightarrow elem(Seq2, N) = class(elem(Seq1, N), Part)$$

At this point, the only two representations that have been explicitly referenced are sequence of identities ('idents') and the Dayhoff transformation ('dayhoff'). They are placed directly in the 'order' relation by the logic about what should be better for homology modeling, so the Dayhoff partition will automatically be selected as the only maximal representation in the first iteration of search. However, additional knowledge suggests that some slight variations of this partition might be better. We define a variant of a partition as another partition in which the class membership of exactly one amino acid is different between them.

$$variant(P, Q) \rightarrow \forall_{A,B}$$

$$(class(A, P) = class(A, Q) \vee class(B, P) = class(B, Q) \vee A = B)$$

These variants are generated as new alternatives for each previous representation that has been evaluated (recorded in the 'state' predicate). They are considered to be potentially better for homology modeling than the representations on which they are based, and are placed appropriately in the partial order.

$$state(P, A)\ \wedge\ variant(Q, P)\ \wedge\ \neg\exists_B\ state(Q, B) \rightarrow order(Q, P)$$

There is no explicit preference among arbitrary variants of the same base representation; all neighbors are assumed to be equivalent. However, we can add one more expectation about the utility of these proposed alternatives that will help guide the search: for any two representations that have already been evaluated, any variant of the one with the higher accuracy is likely to be better than any variant of the one with the lower accuracy. These statements use accuracies recorded in the 'state()' predicate to make the choices during search more intelligent by being dynamic.

$$state(P,A) \;\wedge\; state(Q,B) \;\wedge (A > B) \;\wedge\; variant(X,P) \;\wedge\; variant(Y,Q)$$
$$\rightarrow order(X,Y)$$

6.3.3 Method

The domain knowledge behind this experiment suggests that we should conduct a search through the space of partitions of amino acids, which stand for alternative representations because they can be used to transform examples. In this space, partitions similar to the Dayhoff partition are particularly expected to improve homology modeling, so the search should explore the space in the area around this node. The logical statements constituting the domain theory above could be given to a CORIL-like system to generate a series of representations to evaluate by inferring (via automated deduction) maximal representations in the partial order at each step. However, for historical reasons, we have implemented a procedure in LISP that carries out the same search as is implied by the knowledge.

For the first iteration, the only representation that is explicitly believed to be better for homology modeling than amino acid identities is the Dayhoff partition, so it is naturally chosen as the starting node for the search. Thereafter, in each iteration, the knowledge implies that all unevaluated variants of the best representation evaluated so far will be maximal. They are maximal because they dominate (in the partial order) variants of representations with lower accuracies. Furthermore, no particular variant of the best node is expected to be better than the others. So an equivalent search procedure would be to generate and test random variants of the best node so far.

This search procedure is effectively a Monte Carlo version of hill-climbing (see Figure 6.2). The search operator generates a random swap for the best partition so far by choosing a random amino acid and a random group (not including its current group, but including an empty group

```
(defun search-1 (starting-partition number-iterations)
  (let* ((best-partition starting-partition)
         (best-accuracy (evaluate-representation best-partition)))
    (dotimes (i number-iterations)
      (let* ((variant (random-variant best-partition))
             (accuracy (evaluate-representation variant)))
        (if (> accuracy best-accuracy)
          (setf best-partition variant best-accuracy accuracy))))
    best-partition))

(defun random-variant (partition)
  (let* ((variant (append nil partition))
         (aa (nth (random 20) '(ala cys asp glu phe gly his ile lys leu
                               met asn pro gln arg ser thr val trp tyr)))
         (group (random (length variant))))
    (setf variant (recursive-delete aa variant))
    (setf variant (replace-nth group variant (cons aa (nth group variant))))
    (setf variant (remove-if #'null variant))
    variant))
```

Figure 6.2: Procedural implementation of the search for variant partitions implied by the knowledge involved in this experiment. In this LISP code, partitions are assumed to be represented as lists of lists of amino acid symbols. The input to the search procedure is an initial partition, such as the Dayhoff partition. After hill-climbing for the requested number of iterations, the best variant partition that can be found is returned. The **random-variant** procedure generates alternative partitions by swapping random amino acids to other classes (possibly a new, empty class). The **evaluate-representation** procedure (not shown) will take care of transforming example sequences according to a candidate partition, dividing them into training and testing sets, and running the homology modeling algorithm on them to compute an accuracy estimate.

for allowing the formation of new singleton classes) and re-maps the selected amino acid to the new group. Since the class of only one amino acid has changed, this swapping operation generates variant partitions, as defined in the knowledge above. So the procedure simulates the decisions that a theorem-prover would make in trying to deduce maximal representations.

We ran this search procedure for 10, 20, 50, 100, and 200 iterations (5 minutes to 1.5 hours of runtime on the CM5). In this experiment, the measure of the performance of homology modeling that we used was *accuracy*, which is defined as follows. Given a set of protein sequences, the accuracy of homology modeling is the frequency with which the closest protein to each protein has the same fold class, where closeness is determined by maximal sequence alignment score.

Because of the randomness in the search operator, we ran 10 independent searches for each data point, resulting in 10 different paths of incremental changes to the Dayhoff partition. Furthermore, to ensure that the results were not overfit to the data, we performed 10-fold cross-validation. We randomly divided the set of proteins into 10 balanced subsets (folds). For the first subset, the other 90% of the data was used for training (i.e. to compute the accuracy of homology modeling to guide the hill-climber), and then the resulting accuracy was computed over the test set (i.e. the 10% withheld). This process was repeated for each of the 10 folds, and the average accuracy was reported. So each data point represents the average accuracy for 100 partitions from 10 independent search paths for each of the 10 different training sets.

6.3.4 Results

Figure 6.3 shows the results of this hill-climbing experiment.[3] The starting node for the search (i.e. at 0 iterations) was the Dayhoff partition.[4] When used to re-represent amino acid sequences for homology modeling, this partition results in an accuracy of 56.4%. For comparison, the accuracy of homology modeling when proteins are described in their original representation—amino acid identities—is 77.7% (shown by the horizontal line in the graph).

So the Dayhoff partition itself is worse for homology modeling than the original representation. However, by incrementally searching though the space of variant partitions, better representations were found. After 100 iterations of swapping amino acids among classes, the accuracy was increased to 74.8% on average, bringing the performance of homology modeling nearly back to the baseline for amino acid identities. The improvement leveled off at 100 iterations of search; no further increases in accuracy were achieved by up to 200 iterations (data point not shown). After 100 iterations, the accuracies among the 100 iterations (from 10 runs over 10-fold cross-validation) ranged from 72.0% to 81.0%. The best eleven partitions found are listed in Table 6.4.

The highest accuracy achieved by any partition when used to transform amino acid sequences for homology modeling was 81.0%, as measured over the entire dataset, which was over

[3] This graph was reported in (Ioerger *et al.*, 1995). The data points were generated using a 179-protein subset of the dataset described in Table 6.3. Also, the gap weights used in the alignment algorithm were 3.0 for gap-open-penalty and 0.1 for gap-extend-penalty. These differences do not significantly affect the qualitative nature of the results.

[4] In this experiment, cysteine was initially grouped with the small residues, based on its occasional substitution with other members of this exchange group (Dayhoff *et al.*, 1972).

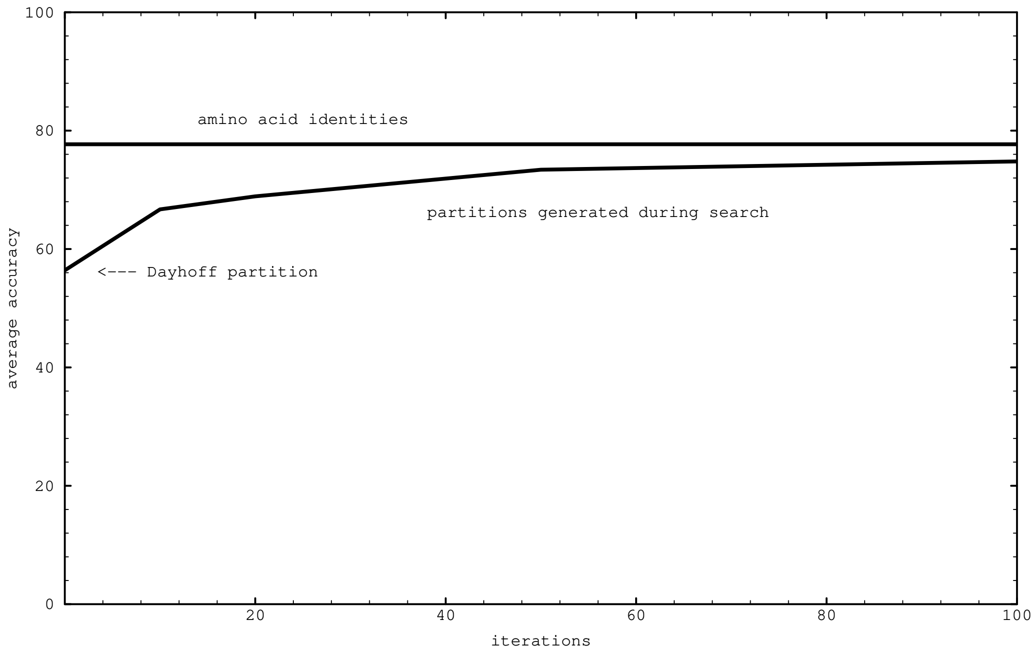

Figure 6.3: Average improvement in the accuracy of homology modeling by incrementally perturbing the Dayhoff partition. The Dayhoff partition (at iteration 0) is significantly improved during search, though the average accuracy never surpasses the baseline for identities (indicated by the horizontal line). Each data point (at 10, 20, 50, and 100 iterations) constitutes the average accuracy (frequency of predicting correct fold classes by analogy to most similar other protein based on sequence alignment score) for partitions improved by iterating the hill-climbing procedure involving random swaps of amino acids to new classes. The accuracies are averages for 10 independent search paths for each fold in 10-fold cross-validation. The standard errors were all less than 0.5%, so error bars are not shown.

Table 6.4: Best eleven partitions found after 100 iterations of the hill-climbing to improve the Dayhoff partition. The amino acids are given by their one-letter codes. Accuracies were computed over the entire dataset.

accuracy	partition
81.0	{{MILV}{KRF}{APT}{CQN}{SW}{Y}{D}{E}{G}{H}}
80.5	{{MILV}{FWY}{HS}{AT}{EQ}{N}{G}{K}{CP}{R}{D}}
79.9	{{MILV}{FWY}{HKP}{DNST}{A}{E}{GQ}{C}{R}}
79.9	{{MILVW}{CF}{KQRY}{HNST}{P}{E}{AG}{D}}
79.9	{{MILV}{FWY}{KQR}{ADNST}{P}{C}{EG}{H}}
79.3	{{MILV}{NWY}{HR}{AKS}{DEQ}{G}{C}{F}{T}{P}}
79.3	{{MILV}{FWY}{HKPR}{EST}{GN}{C}{A}{D}{Q}}
79.3	{{MILVW}{Y}{AHKR}{DNST}{E}{G}{F}{P}{C}{Q}}
79.3	{{MILV}{FWY}{HKR}{ADNQST}{P}{G}{E}{C}}
79.3	{{MILV}{FY}{HKQR}{AENPST}{D}{G}{C}{W}}
79.3	{{MIL}{FWY}{DKR}{AEST}{H}{G}{C}{N}{Q}{P}{V}}

3% better than amino acid identities. However, because this representation was selected based on sequential tests over the same data, we cannot be sure that it *individually* is not overfit, producing an artificially high score by tuning to the set of data used in the experiment. We can only be sure that, by 100 iterations, this search is likley to find representations with an accuracy of 74.8% *on average*. The validity of particular representations that are discovered and appear to be good must be ultimately be evaluated on other sets of proteins.

6.3.5 Conclusion

The Dayhoff partition turned out to be worse for homology modeling than amino acid identities (56.4% vs. 77.7%). Furthermore, on average the search did not find representation (alternative partitions) that were better for homology modeling than identities. However, this experiment shows the ability of empirical search in conjunction with domain knowledge to improve *some* representation—in this case, the Dayhoff partition. The method of evaluating individual changes in representation by their effect on the performance of homology modeling as a learning algorithm led to the discovery of representations better than the Dayhoff partition by incrementally accumulating improvements.

6.3.6 Analysis

Among the partition in Table 6.4, several patterns in the perturbed groupings of amino acids may be observed. First, certain groups remained fairly stable during search, such as the hydrophobic group, which appears unchanged in six of the eleven best partitions, and appears only slightly modified in the rest. On the other hand, other groups have become significantly altered. For example, the small group, which initially contained 10 residues, seems to have become consistently fractured into smaller groups during search.

One way to analyze these patterns is to examine the frequency with which each pair of amino acids is found grouped together in improved partitions. The swapping operator tends to split amino acids apart, so if they are found together frequently after many iterations, this might indicate some advantage for homology modeling.

As a set of partitions for tabulating these frequencies, we used 100 partitions resulting from independently improving the Dayhoff partition for 100 iterations. The accuracies of these partitions ranged from 72% to 81%. For each pair of amino acids, we counted the number of partitions in which both amino acids were grouped in the same class. Table 6.5 presents the 25 most frequently co-occurring pairs of amino acids. Of the 190 possible pairs, these pairs occurred in at least 40% of the partitions. The rest of the pairs of amino acids associated together with frequencies approximately normally distributed below this point.

These pairings form four distinct clusters (see Figure 6.4). Three of these clusters correspond directly to Dayhoff exchange groups: hydrophobics, aromatics, and positives. Each pair within these groups was found to be frequently associated in the perturbed partitions, and no significant associations with amino acids in other groups was observed.

These clusters help resolve certain group memberships (and hence property definitions) that have previously been debated. For example, Taylor's (1986) definition of amino acid properties differs slightly from Dayhoff's (1972). Taylor suggests that methionine might not belong to the core group of hydrophobics because it has a polar sulfer atom, and he suggests that histidine might belong to the group of aromatics due to its conjugated ring. Our analysis confirms Dayhoff's groupings (methionine with the hydrophobics, histidine with the positives), which appeared to be best for the purposes of homology modeling.

Table 6.5: Most frequently associated pairs of amino acids in the 100 perturbed partitions. These 25 of 190 possible pairs occurred in the same class in at least 40 of the partitions, suggesting their utility for homology modeling.

pair	count	pair	count	pair	count
Ile-Val	98	His-Arg	60	Ala-Ser	46
Ile-Leu	96	Trp-Tyr	60	Glu-Asn	46
Leu-Val	94	Asn-Thr	60	Asp-Ser	45
Ile-Met	92	Phe-Trp	55	His-Lys	44
Met-Val	90	Ser-Thr	53	Ala-Asn	44
Leu-Met	88	Asn-Ser	52	Glu-Thr	43
Phe-Tyr	67	Asp-Glu	51	Ala-Asp	42
Asp-Asn	64	Asp-Thr	48		
Lys-Arg	61	Ala-Thr	47		

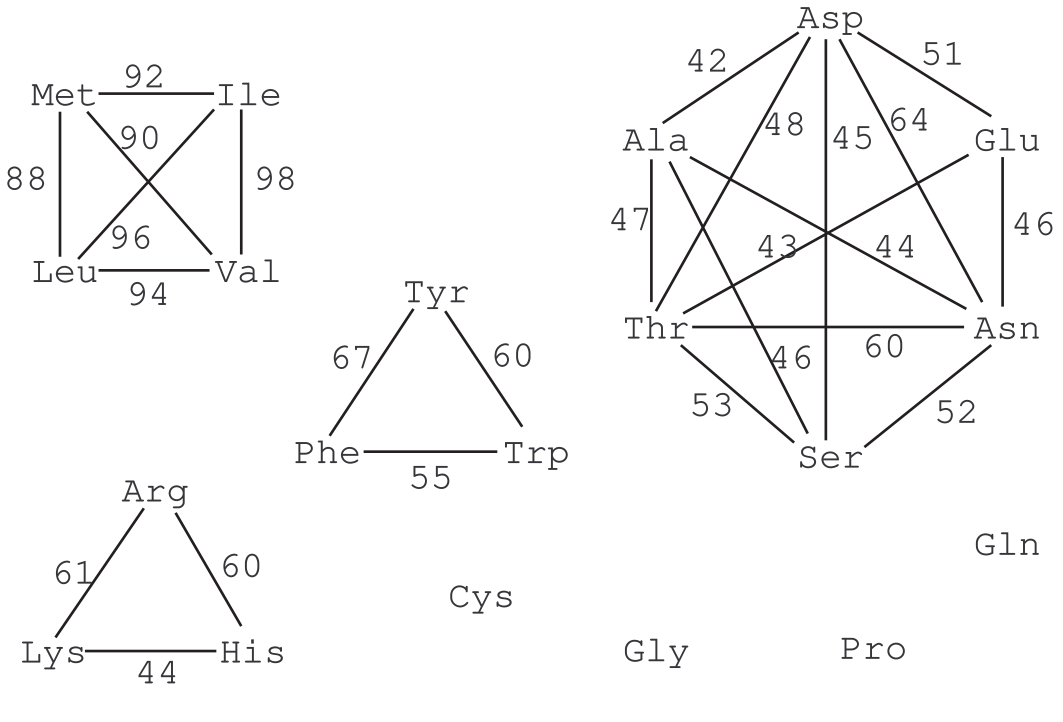

Figure 6.4: Pair association clusters. A link is drawn for each pair of amino acids listed in Table 6.5 (most frequently associated in perturbed partitions). These associations fall into four well-defined clusters, with four additional residues (Cys, Gly, Pro, and Gln) forming their own individual clusters, unlinked to anything else.

Although pairs of amino acids within these clusters tended to group together, the frequencies (marked on the links in Figure 6.4) show variable levels of strength. The hydrophobic residues were almost never split apart; the weakest pair, methionine and leucine, were only separated in 12% of the partitions. This indicates the prime importance of this property, which is consistent with studies such as by Kidera et al. (1985). The aromatic residues were found grouped together in fewer partitions (55-67%), and positives were grouped together even less frequently (44-61%). This may indicate problems with the definitions of these properties, or perhaps weaker relevance for protein folding.

The fourth cluster resembled the small group of amino acids, except cysteine, glycine, proline, and glutamine were split out into their own singleton clusters. Of the 15 possible pairwise relationships among the six residues left in this cluster, 13 pairs were found to be associated in the same class in at least 40% of the partitions, making the cluster well-defined with respect to transitivity. By revising the Dayhoff partition based on this analysis (reducing the small group to {A,D,E,N,S,T}, and making C, G, P, and Q unique), the accuracy of homology modeling is increased to 77.1%, which is almost identical to the baseline performance of homology modeling with the original representation in terms of amino acid identities (77.7%).

Of the four residues split out of the original small group, three of them are easily explained, and one is surprising. Cysteine is unique because it is the only residue that can form disulfide bonds to cross-link a protein structure. Glycine is the only residue without a beta-carbon, allowing it to assume left-handed conformations. And proline is the only residue whose side-chain reconnects to the main-chain, producing an extra rigidity in the backbone. However, it is not clear why glutamine does not frequently associate with any other residue. Richardson and Richardson (1989) comment that glutamine is one of the most unremarkable amino acids because it does not appear to be able to play any unique roles in protein structure. Our analysis suggests that the role of glutamine needs to be re-examined.

This analysis has shown a domain-specific way of summarizing the results of search for alternative representations to extract refinements of domain knowledge. Our analysis of the frequencies of co-occurrence for pairs of amino acids indirectly evaluates the definitions of amino acid properties by seeing how consistent they are with patterns in alternative representations that turned out to be good. This feedback is made possible because suggestions for alternative representations depended on knowledge of amino acid properties, in the sense that partitions similar to the Dayhoff partition were expected to be good. Our analysis has confirmed the membership of amino acids in certain groups (e.g. methionine and histidine), ranked properties by apparent strength (hydrophobic > aromatic > positive), suggested a revised definition for one group (small={A, D, E, N, S, T}), and pointed out an anomalous amino acid that needs to be studied further (glutamine).

6.3.7 Possible Extensions of this Experiment

There are several ways this experiment could have been done differently. First, instead of doing Monte Carlo hill-climbing, we could evaluate every possible variant at each step. However, given 20 amino acids and 5 groups, there are 100 swaps to evaluate before a single step could be taken (already as many iterations as in the experiment described). The advantage of this approach would be that it would exhaustively find the best swaps at each stage, perhaps requiring only a few steps to optimize the partition.

Another alternative would be to perform a kind of simulated annealing in which some swaps that decreased accuracy only slightly would still be accepted in hill-climbing. This would allow some backward steps that do not increase the accuracy of homology modeling, but may be necessary if multiple swaps need to be coordinated. For example, suppose that it was important that two amino acids were grouped together (e.g. aspartate and glutamate), but they were initially in a sub-optimal group (e.g. SMALL versus CHARGED). Single-residue swaps would only decrease accuracy by splitting them apart; yet simulated annealing might temporarily allow this, so that later the two residues can be rejoined in a better group. Another similar variation would be to carry out a parallel search, for example by using techniques such as genetic algorithms (Holland, 1975). Partitions could be encoded in a special language as bit-vectors which allow cross-overs to be defined. Individual groups could then be optimized is separate partitions and recombined later.

Finally, we could encode a more intelligent swapping operator that incorporates more specific domain knowledge. Currently, the random swaps made during search are often illogical because they move amino acids into groups with which they share no obvious properties. Ideally, we would like to focus the search by trying swaps of particular interest, such as grouping histidine with the aromatic residues versus the positive group, or trying glycine in its own group since it is expected to be fairly unique. These suggestions could be derived from knowledge about secondary or plausible properties, which could then help determine which swaps are likely to be useful *a priori*. Further, we could add some rules that compute the expected utility of each swap given the previously evaluated partitions (via accuracies in the 'state()' predicate). This would allow the system to dynamically adjust its decision-making, eventually choosing swaps that have historically helped other partitions.

6.4 Experiment 2

6.4.1 Purpose

Experiment 1 appeared unable to surpass the baseline performance (of homology modeling with amino acid identities) on average. A possible explanation for this is that the partition formalism itself limits expressiveness. Each amino acid has multiple properties, and these define overlapping dimensions of similarity, which cannot be simultaneously represented in a partition (each amino acid is forced to belong to only one group). However, it is believed that the property that is most relevant at a given site depends on context, such as the tertiary environment (Overington *et al.*, 1992). This observation suggests that we could maintain different partitions for different contexts. In this experiment, we explore a sub-space of alternative representations in which amino acids are mapped to property names, but this is done differently in different contexts. Thus the space of representations will be parameterized by *sets* of partitions (one for each context), rather than just a single partition of amino acids. The search will involve finding the best properties for each context, which is not known *a priori*.

6.4.2 Background Knowledge

In this section, we formally describe the background knowledge of biophysics that drives this experiment. The statements fall into two sets. The first set deals with establishing the utility of context-dependent transformations: 1) a property-based re-representation should help homology modeling (as with experiment 1), 2) transforming amino acids into properties is a function of context, which depends on the tertiary environment, and 3) the tertiary environment can be approximated by hydrophobicity patterns among sequential neighbors of each site.

The second set of statements is concerned with suggesting which of these alternative representations is expected to give the largest increase in the accuracy of homology modeling: 1) for any known representation, a variant in which amino acids in some new context are mapped together when they share some interesting property might help, 2) the effects of selecting properties in different contexts should be independent, and 3) when selecting some property in some context makes an improvement, this is evidence that selecting other properties in that context might help. We present a logical formalization of this knowledge that could be given to a theorem-proving system like CORIL to generate alternative representations to search by inference (i.e. automated deduction).

First, the following rules about sequences of properties being a better representation of proteins for homology modeling than amino acid identities carries over directly from the first experiment.

$$\begin{aligned}
&\forall_{X,Y}\ protein(X)\ \wedge\ protein(Y)\ \wedge\ fold(X) = fold(Y)\\
&\quad \rightarrow similar(struct3D(X), struct3D(Y))\\
&\quad \rightarrow similar(seq(X, localStructs), seq(Y, localStructs))\\
&\quad \rightarrow similar(seq(X, roles), seq(Y, roles))\\
&\quad \rightarrow similar(seq(X, props), seq(Y, props))
\end{aligned}$$

In this experiment, however, the transformation of the amino acid at a site into its properties depends on its context. To effect this, we assume that the class of an amino acid (in the 'class' predicate) is a function of its context, which we assume is defined by its tertiary environment. This is a revision of the transformation rule used in the first experiment.

$$\begin{aligned}
&similar(seq(X, props), seq(Y, props))\\
&\quad \rightarrow similar(transform(seq(X, idents), props), transform(seq(Y, idents), props))\\
&transform(seq(X, idents), P) = Y\ \wedge\ (1 < N < length(X))\\
&\quad \wedge\ element(seq(X, idents), N) = A\ \wedge\ context(X, N) = C\\
&\quad \wedge\ class(A, C, P) = B \rightarrow elem(Seq2, N) = B\\
&elem(seq(X, tertEnv), N) = C\ \wedge\ (1 < N < length(X)) \rightarrow context(X, N) = C
\end{aligned}$$

We do not know have access to the tertiary environment at each site, but we do know that some sequential neighbors are likely to be involved. In general, the side-chains of the immediate neighbors ($i \pm 1$) are probably nearby. For residues in a beta conformation ($(\phi, \psi) = (-90, +120)$), the side-chains of $i \pm 2$ are adjacent to the central residue. Finally, for the other major sub-class of sites, the alpha conformation ($(\phi, \psi) = (-60, -60)$), contact is often made with side-chains at $i \pm 3$. So, residues $i - 3 \ldots i + 3$ are likely to form some part of the tertiary environment. Strictly, there is a disjunction of contacts (since any given site is either alpha, beta, or other), but we add a rule that brings in all possible contacts based on potential conformations as part of the tertiary environment.

$$seq(S, conformation) \wedge (1 < N < length(S)) \wedge elem(S, N) = alpha$$
$$\rightarrow contact(N-3, N) \wedge contact(N+3, N)$$
$$seq(S, conformation) \wedge 1 < N < length(S) \wedge elem(S, N) = beta$$
$$\rightarrow contact(N-2, N) \wedge contact(N+2, N)$$
$$seq(S, conformation) \wedge 1 < N < length(S) \wedge elem(S, N) = other$$
$$\rightarrow contact(N-1, N) \wedge contact(N+1, N)$$
$$\exists_C\ (elem(seq(A, conformation), N) = C \wedge (1 < N < length(A)) \rightarrow contact(X, N))$$
$$\rightarrow \forall_Y\ partOfTertEnv(Y + (X - N), Y)$$

Given knowledge of which neighboring amino acids might be participating in the tertiary environment, we can distinguish various contexts based on differences among these residues that are expected be relevant to determining the local structure at the central site. Since hydrophobicity is known to be major influence on protein structure (Schulz and Schirmer, 1979), we can use the pattern of hydrophobicity among these six neighbors to define a notion of context. A simple definition of hydrophobicity (more relaxed than Dayhoff's (1972)) is to divide the amino acids into two subsets: those with polar atoms (hydrophilic: C, D, E, G, H, K, N, Q, R, S, T, Y) and those without (hydrophobic: A, F, I, L, M, P, V, W).[5] So 64 distinct contexts are generated, differing in hydrophobicity (one of two values) in at least one of the six sequential neighboring positions.

[5] There are some exceptions: methionine and tryptophan have weakly polar atoms, but are large enough that their hydrophobicity dominates. Also, glycine has no side-chain at all, but due to the exposure of the carbonyl in its backbone, it often acts hydrophilic (Richardson and Richardson, 1989).

$$primaryEffectOnStructure(hydrophobicity)$$

$$value(hydrophobicity, ala, hydrophobic)$$

$$value(hydrophobicity, cys, hydrophilic)$$

$$\ldots$$

$$\begin{aligned}
&partOfTertEnv(X, N) \wedge primaryEffectOnStructure(H) \\
&\quad \wedge (1 < N < length(S)) \wedge elem(seq(S, idents), N) = A \\
&\quad \wedge (1 < X < length(T)) \wedge elem(seq(T, idents), X) = B \\
&\quad \wedge element(seq(S, tertEnv), N) = C \wedge elem(seq(T, tertEnv), N) = D \\
&\quad \wedge value(H, A, H1) \wedge value(H, B, H2) \wedge H1 \neq H2 \rightarrow C \neq D
\end{aligned}$$

This notion of context can be used to select which properties are used to transform each site. Since a single property is anticipated to be dominant at each site, they are not constrained to fit together in the same partition; the rest of the amino acids (which don't have the property selected for a given context) can remain in their own singleton classes. Thus we propose an expanded list of 18 plausible properties (see Table 6.6) in terms of (overlapping) groups of amino acids, most of which are subsets or supersets of Dayhoff's exchange groups (often corresponding to different degrees of a property, as is done by Taylor (1986)).

Therefore the space of alternative representations suggested by this domain knowledge consists of sets of 64 partitions, each of which groups amino acids together in one of 18 pre-defined ways. Given such a specification for an alternative representation, a sequence would be transformed by mapping each amino acid into either the symbol for the property specific to its particular context, or to its own identity if it does not have that property. The selection of the appropriate partition is done by 'class' predicate, which has context as an argument. The specific representation referred to above as 'props' exists as one of the alternatives in this space.

$$propList(coreHydrophobic, [Met, Ile, Leu, Val])$$

$$propList(largeHydrophobic, [Met, Ile, Leu, Val, Phe, Trp])$$

$$\ldots$$

$$\begin{aligned}
&\forall_C \exists_P \; propList(P, L) \\
&\quad \wedge (member(A, L) \rightarrow class(A, C, props) = P) \\
&\quad \wedge (\neg member(A, L) \rightarrow class(A, C, props) = A)
\end{aligned}$$

However, we have not yet suggested in a constructive sense which of these alternatives is expected to be better for homology modeling than identities. Consider the 'null' representation

Table 6.6: Extended list of properties used in experiment 2. Each property is defined by the subset of amino acids that have that property. These properties are generally plausible variations of Dayhoff's exchange groups (Table 6.2; Dayhoff et al., 1972).

property	*amino acids*
core-hydrophobic	met ile leu val
large-hydrophobic	met ile leu val phe trp
small-hydrophobic	ala pro met ile leu val
all-hydrophobic	ala pro met ile leu val phe trp
aromatic	phe trp tyr
left-handed	gly asn
tiny	ala cys gly ser
hydroxyl	ser thr
carboxyl	asp glu
amide	asn gln
carbonyl	asp gln asn gln
positive	his arg lys
positive-no-his	arg lys
charged	his lys arg asp glu
small-polar	ser thr asn gln
small-strong-polar	ser thr asp glu asn gln
strongly-polar	his arg lys asp glu asn gln
all-polar	his arg lys ser thr asp glu asn gln

in which each amino acid in each context is mapped to itself, regardless of context. It expresses no properties, and is equivalent to the original representation in terms of amino acid identities. We expect that if we group together some amino acids that share a property in some context in this initial representation, then it might be better for homology modeling.

Based on this idea, we propose an operator that defines variants (neighboring representations) of a given representation by mapping all amino acids the same way, with the following exceptions: in some context for which no property has been selected (all amino acids currently in singleton classes), the subset of amino acids having one of the 18 properties listed in Table 6.6 are mapped to the symbol for that property. We use 'variant(P,Q,C,D)' to mean that representation P is the variant of representation Q obtained by mapping all amino acids with property C to the symbol C in context D.[6] Any such (unevaluated) variant of a known representation is expected to be better than the representation it is based on, which finally connects to the 'order' predicate.

$$\begin{aligned} & variant(P,Q,C,D) \ \wedge \ class(A,E,P) = J \ \wedge \ class(A,E,Q) = K \ \wedge \ (J \neq K) \\ & \quad \rightarrow (E = D) \ \wedge \ J = C \ \wedge \ K = A \ \wedge (propList(C,L) \rightarrow member(K,L)) \\ & state(S,A) \ \wedge \ variant(V,S,B,C) \ \wedge \ \neg\exists_D \ state(V,D) \rightarrow order(V,S) \end{aligned}$$

Thus at any stage, all unexplored variants (neighbors) of all representations evaluated so far are candidates for being selected in the search (since they are maximal in the partial order). We have some additional knowledge that can give preferences among these alternatives. First, we expect that the effect of selecting properties in different contexts will be independent, since they will be applied to different sites in proteins. One way to express this knowledge is to claim that, given two representations whose accuracies are known, making a particular variant of the representation with higher accuracy should be better than making the same variant of the representations with lower accuracy.

$$\begin{aligned} & state(S1,A1) \ \wedge \ state(S2,A2) \ \wedge \ A1 > A2 \ \wedge \ variant(V1,S1,X,Y) \\ & \quad \wedge \ variant(V2,S2,X,Y) \ \wedge \ \neg\exists_{B1} \ state(V1,B1) \ \wedge \ \neg\exists_{B2} \ state(V2,B2) \\ & \quad \rightarrow order(V1,V2) \end{aligned}$$

The effect of this constraint is that, once an improvement is made in one context, no variants in other contexts need to be explored for the base representation. Exploring other contexts may

[6] Note that this definition implicitly enforces the constraint that no context can have two or more properties selected (they might overlap; only one should be playing a primary role).

be done with the improved representation. This eliminates many redundant tests that would eventually arrive at the same representation by different paths (same choices of properties in contexts, but selected in different orders).

Thus representations can be optimized one context at a time, and we can build representations incrementally by accumulating changes (like hill-climbing). There is one more piece of knowledge that provides even more guidance in the search. Suppose a property is found to improve homology modeling when selected for a given context. Now there is evidence that this context makes a difference; other contexts may not have much impact if they do not occur often in proteins. So this last argument places a constraint (via the 'order' predicate) that unexplored variants in a context for which some property in known to improve homology modeling should be preferred over variants of other contexts (which have not yet been explored at all).

$$
\begin{aligned}
&state(S1, A1) \wedge state(S2, A2) \wedge variant(V1, S1, W, X) \\
&\quad \wedge variant(V2, S2, Y, Z) \wedge \neg\exists_{B1}\, state(V1, B1) \wedge \neg\exists_{B2}\, state(V2, B2) \\
&\quad \wedge (\exists_{P1}\, variant(P1, S1, G, X) \wedge state(P1, A3) \wedge A3 > A1) \\
&\quad \wedge (\neg\exists_{P2}\, variant(P2, S2, H, Z) \wedge state(P2, A4) \wedge A4 > A2) \\
&\quad \rightarrow order(V1, V2)
\end{aligned}
$$

So this knowledge specifically motivates staying at a node until all properties in one context have been tried. Thereafter, the search will proceed to the best variant, accepting the change as permanent, and begin to explore properties in another context. Because the definition of variants precludes multiple properties from being selected into the same context, none of the variants in the first context will ever be tried again. This produces a very focused search; only 18×64 possibilities are evaluated, out of a space of 18^{64} possibilities, taking 10 hours of CPU time on the CM5. Without these constraints, the search would be infeasible.

6.4.3 Method

A system like CORIL could use this knowledge to generate a search by automated deduction, inferring representations to evaluate at each stage by solving for maximal nodes in the partial order. However, we designed a LISP implementation that simulates the same decision procedure as implied by the knowledge.

Contexts are represented by integers between 0 and 63. They are computed by determining the hydrophobicity of the six neighboring amino acids, three on either side, of a given site,

```
(defun context (seq pos)
  (let ((len (length seq)))
    (if (or (< pos 3) (>= pos (- len 3)))
      -1 ; context undefined near terminals
      (let ((a (hydrophobicity (nth (- pos 3) seq)))
            (b (hydrophobicity (nth (- pos 2) seq)))
            (c (hydrophobicity (nth (- pos 1) seq)))
            (d (hydrophobicity (nth (+ pos 1) seq)))
            (e (hydrophobicity (nth (+ pos 2) seq)))
            (f (hydrophobicity (nth (+ pos 3) seq))))
        (+ a (* 2 (+ b (* 2 (+ c (* 2 (+ d (* 2 (+ e (* 2 f)))))))))))))
```

Figure 6.5: Procedure for computing contexts. Inputs are an amino acid sequence (seq) and an integer position (pos). Output is an integer 0..63 for the context at that position in the sequence. The function **hydrophobicity** returns 0 if an amino acid is hydrophobic, and 1 if it is hydrophilic. If the context is undefined (i.e. near the termini of the sequence), then -1 is returned.

encoding this as a bit-string, and interpreting it as a binary number. For sites that are so near the terminals of a protein that some of their neighbors are not defined, we assign them to a default context (-1) in which amino acids are never merged together. Figure 6.5 gives the LISP code for computing the context at a given site in a given amino acid sequence.

In this implementation, a representation is encoded by a list of 64 partitions, which are themselves lists of lists of symbols for amino acids, as in experiment 1. Figure 6.6 shows how, given such a representation and a sequence, the sequence may be transformed by replacing the amino acid at each site with a symbol representing its class in the partition specific for its context.

In this experiment, the starting node for the search is the 'null' partition, which is equivalent to the identity representation (all amino acids mapped to themselves). The search procedure, shown in Figure 6.7, starts by merging the amino acids together in first context that share the first property in Table 6.6. It evaluates this alternative, calling a routine that takes care of applying the transformation to some example sequences, dividing them into training and testing sets, and calling the homology modeling routine to get an estimate of the effect of the change on the inductive performance.

This process is repeated independently with the rest of the 18 properties in the first context. The property that gives the highest increase in accuracy is accepted as a permanent change,

```
(defun transform (seq repr)
  (let ((newseq nil))
    (dotimes (i (length seq))
      (setf newseq
        (cons (class (nth i seq) (nth (context seq i) repr)) newseq)))
    (reverse newseq)))

(defun class (amino-acid partition)
  (assoc (invert-partition partition) amino-acid))
```

Figure 6.6: Procedure for transforming sequences. Inputs are an amino acid sequence (seq) and an alternative representation (repr), which is a list of 64 partitions of the 20 amino acids. Output is a new sequence of the same length but with integers for elements. At each site, the context is computed and used to select the appropriate partition from the set of 64. Then the class name of the amino acid identity at that site is looked up in the selected partition using the **class** function, which inverts a partition to generate integer class names for each amino acid.

and the routine proceeds to evaluate each of the properties in the next context. This procedure iterates through all 64 contexts and returns the final representation, with the best property merged in each context. Note that the knowledge does not give any preferences about what order the contexts should be explored; our procedure happens to explore them in numerical order.

In these experiments we used the *average z-score* (defined below) as a measure of the inductive performance (predictive ability) of homology modeling, rather than accuracy, as used in experiment 1. Since accuracy is defined as the frequency of correct fold-classification, where each observation is binary (right or wrong), the precision of estimates is limited to $1/N$, where N is the size of the dataset. Since we have around 200 sequences, we cannot detect differences between representations that cause a change of less than about 0.5% in accuracy. All the representations in this experiment (along with the substitution table method described below) differ by less than 1% in accuracy, which effectively makes the results depend on changing the classification of just one or two proteins.

However, molecular biologists often use z-scores as a more refined estimate of the relationship between two proteins. The z-score of an alignment is interesting to molecular biologists because it indicates the *structural significance* of any apparent sequence similarity. A z-score is computed by comparing an alignment score S between two proteins to a background distri-

```
(defun search-2 ()
  (let ((repr (identity-representation)))
    (dotimes (context 64)
      (let ((best-variant nil) (best-accuracy 0))
        (dolist (property interesting-properties)
          (let* ((variant (construct-variant repr context property))
                 (accuracy (evaluate variant)))
            (if (> accuracy best-accuracy)
              (setf best-variant variant best-accuracy accuracy))))
        (setf repr best-variant)))
    repr))

(defun identity-representation ()
  (let ((partitions nil))
    (dotimes (i 64) (setf partitions (cons singleton-partition partitions)))
    partitions))

(setf singleton-partition
  '((ala)(cys)(asp)(glu)(phe)(gly)(his)(ile)(lys)(leu)
    (met)(asn)(pro)(gln)(arg)(ser)(thr)(val)(trp)(tyr)))

(setf interesting-properties
  '((core-hydrophobic (met ile leu val))
    (large-hydrophobic (met ile leu val phe trp)) ...))

(defun construct-variant (repr context property)
  (replace-nth context repr
    (merge-amino-acids (nth context repr) (second property))))

(defun merge-amino-acids (partition amino-acids)
  (let ((newpart partition))
    (dolist (aa amino-acids) (recursive-delete aa newpart))
    (cons amino-acids newpart)))
```

Figure 6.7: Search procedure for experiment 2. The search starts with the identity representation, which in the context-dependent formalism corresponds to 64 singleton partitions. The search iterates through each of the 64 contexts, evaluating the effect of merging each interesting group of amino acids and selecting the best. The **construct-variant** procedure takes a set of partitions, a context integer, and a property, and replaces the requested partition (indexed by the context) with a new partition in which the amino acids having the requested property are merged together into the same class. The **evaluate** procedure (not shown) is responsible for applying individual representations to example sequences and running the homology modeling algorithm on them to compute the effect on accuracy.

bution of alignment scores between proteins known not to be in the same fold class. Given the mean M and standard deviation D of this distribution, the z-score Z is computed in the standard statistical way: Z=(S-M)/D. Because the background distribution is of alignments between unrelated proteins, a score significantly above this distribution suggests that the proteins are not unrelated. Hence changes of representation are preferred that increase the z-scores of alignments between proteins that are related. This will make proteins with similar structures easier to recognize, and predictions of such relationships will become more confident. Thus to evaluate a representation, we report the average z-score over pairs of proteins in the same fold class (see Figure 6.8). Since z-scores are continuous, effectively capturing the 'closeness' of a fold-class prediction to being correct, this measure can distinguish smaller differences between representations, and hence provides a more refined assessment of the performance of homology modeling.

We ran this experiment, starting from identities and incrementally selecting the best properties for individual contexts, one at a time. This search involved a total of $18 \times 64 = 1152$ evaluations (taking approximately 10 CPU hours on the CM5). At the end of each stage in the search, we measured the accumulated change in the z-score. There are 944 pairs of proteins in the same fold class in our dataset, with which we can compute the average z-score. However, to avoid overfit, these pairs were randomly divided into two subsets: two-thirds (652) for training, and one-third (292) for testing. The sequence pairs for training were used to compute z-scores during the search for determining which properties were most useful (for homology modeling) in each context. After the search was finished, the sequence pairs for testing were used to re-evaluate the change in z-score for the representation selected at each stage. This approach ensures that any observed improvements in the performance of homology modeling are not specific to the data used for training, and that the utility of these representations generalizes to other proteins.

6.4.4 Results

The results of the search in this experiment are shown in Figure 6.9. The starting node for the search (i.e. iteration 0) corresponded to the identity representation. The average z-score for this baseline representation was 5.22. When this search process had iterated over all 64 contexts, the average z-score had increased to 5.64, a change of +0.42, as computed over the testing set

```lisp
(defun compute-z-score (repr seqs)
  (let* ((transformed (mapcar #'(lambda (x) (transform x repr)) seqs))
         (table (generate-pairwise-scores transformed))
         (len (length seqs))
         (sum 0) (count 0))
    (dotimes (i len)

      ; first loop over other seqs in different fold classes
      (let ((unrelated nil))
        (dotimes (j len)
          (if (not (eq (fold (nth i seqs)) (fold (nth j seqs))))
            (setf unrelated (cons (aref table i j) unrelated))))

        ; compute statistics of background distribution
        (let ((mean (average unrelated))
              (stdev (standard-deviation unrelated)))

          ; for other proteins in same fold class...
          (dotimes (j len)
            (if (and (eq (fold (nth i seqs)) (fold (nth j seqs)))
                     (not (= i j)))

              ; normalize with statistics and add to running total
              (let ((z-score (/ (- (aref table i j) mean) stdev)))
                (setf sum (+ sum z-score))
                (incf count)))))))
    (/ sum count)))

(defun generate-pairwise-scores (sequences)
  (let* ((n (length sequences))
         (table (make-array (list n n) :initial-element 0)))
    (dotimes (i n)
      (dotimes (j n)
        (setf (aref table i j) (align (nth i sequences) (nth j sequences)))))
    table))
```

Figure 6.8: Procedure for computing z-scores for an individual representation. Inputs are the representation (set of 64 partitions: repr), and a set of example protein sequences (seqs). For each sequence, the mean and standard deviation of scores from alignments to proteins in different fold classes are computed. Then these statistics are used to normalize alignment scores to proteins in the same fold class. The average z-score is returned. The **generate-pairwise-scores** function caches alignments scores (from running the sequence alignment algorithm; not shown) in a matrix.

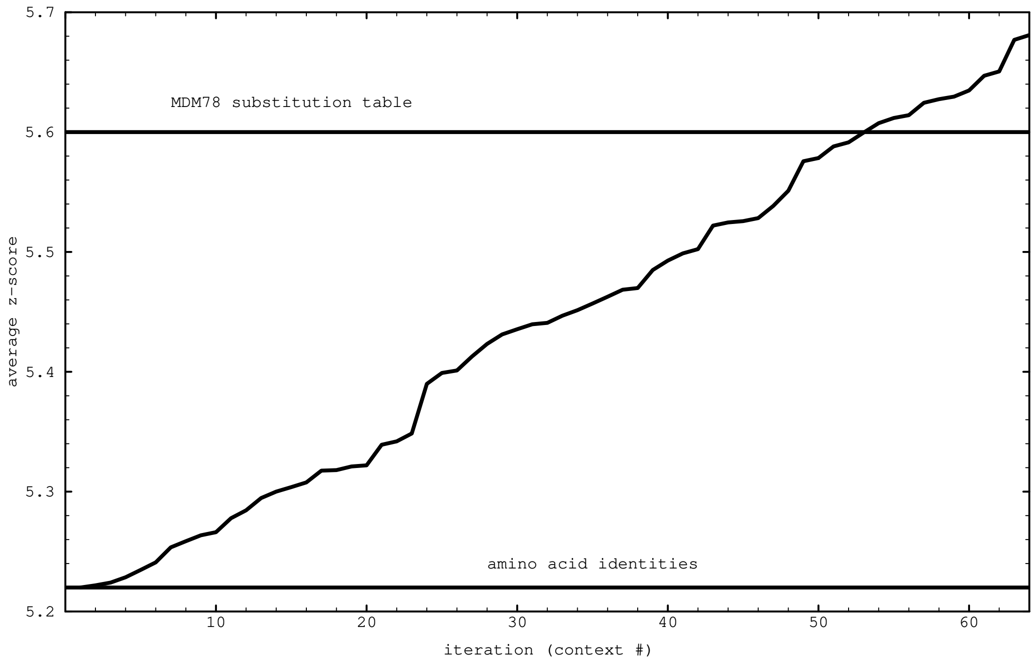

Figure 6.9: Improvement of average z-score during search. In each iteration of the search, a new context was selected, each of the 18 interesting groupings of amino acids in Table 6.6 were formed in the corresponding partition, and the change that gave the best improvement in z-scores was kept. The starting point for the search was equivalent to the original representation in terms of amino acid identities, which had an absolute z-score of 5.22 (lower horizontal line). The upper horizontal line indicates the performance of homology modeling when using the MDM78 substitution table (absolute=5.60, delta=+0.38). These scores were computed over the training data, with the final representation surpassing MDM78 in average z-score (absolute=5.68, delta=+0.46). On separate testing data, the final representation still had a higher z-score (absolute=5.64, delta=+0.42) than MDM78.

(sequence pairs not used during training). This means that alignments of proteins with similar structures are becoming more significant (relative to structurally dissimilar proteins), making it easier to recognize when proteins are in the same fold class. Thus the final representation increases the sensitivity of sequence alignments, and hence increases the ability of homology modeling to predict protein structures. By the end of the search, the curve had not leveled off, and appears able to gain further increases. However, the search was terminated because it was only designed to select one property into each of the 64 contexts. The curve might be extended by allowing the selection of additional properties into each context, though the experiment as performed is already approaching computational bounds (10 hours of CPU time on the CM5).

Table 6.7: Best context-dependent representation found by the search procedure used in experiment 2. The contexts are given as bit-patterns for the six sequential neighbors of a site i: i-3, i-2, i-1, i+1, i+2, i+3. 0 means the neighbor is hydrophobic, and 1 means hydrophilic. To transform a protein sequence into this representation, the context at each site is determined, and then the central amino acid is replaced by either the name of the property for that context if it has the property, or by itself (amino acid identity) otherwise.

context	group	context	group	context	group	context	group
000000	all-hydrophobic	000010	large-hydrophobic	000001	all-polar	000011	core-hydrophobic
100000	small-hydrophobic	100010	core-hydrophobic	100001	all-polar	100011	all-hydrophobic
010000	all-polar	010010	carbonyl	010001	large-hydrophobic	010011	all-hydrophobic
110000	all-polar	110010	all-polar	110001	small-hydrophobic	110011	hydroxyl
001000	all-polar	001010	carbonyl	001001	charged	001011	all-polar
101000	small-hydrophobic	101010	small-strong-polar	101001	all-polar	101011	large-hydrophobic
011000	strongly-polar	011010	all-polar	011001	tiny	011011	positive
111000	small-strong-polar	111010	aromatic	111001	all-polar	111011	strongly-polar
000100	small-hydrophobic	000110	all-polar	000101	small-strong-polar	000111	tiny
100100	all-hydrophobic	100110	all-polar	100101	all-polar	100111	tiny
010100	strongly-polar	010110	small-polar	010101	small-polar	010111	aromatic
110100	all-hydrophobic	110110	large-hydrophobic	110101	small-polar	110111	small-strong-polar
001100	strongly-polar	001110	all-hydrophobic	001101	small-stong-polar	001111	all-polar
101100	all-polar	101110	tiny	101101	small-strong-polar	101111	core-hydrophobic
011100	large-hydrophobic	011110	hydroxyl	011101	large-hydrophobic	011111	small-polar
111100	small-hydrophobic	111110	hydroxyl	111101	core-hydrophobic	111111	aromatic

The best representation discovered by the search is given in Table 6.7, indicating the most useful property found for each context. Of the 18 groups initially described as being potentially relevant, 13 were actually found to be useful in at least one context, spanning a wide range of physical and chemical properties. This observation highlights the importance of taking context into account when re-representing amino acids in terms of properties. Without context, all these properties would have to be collapsed into one universal partition applicable to all sites, making only a few properties (such as hydrophobicity) appear to be important. Context-dependent partitions can capture independent patterns in subsets of sites which might have unique constraints.

To put this increase in z-score in perspective, we computed the average z-score for the best known alternative method for dealing with the hidden-homology problem: substitution tables (Dayhoff *et al.*, 1978). Substitution tables contain numerical estimates for the likelihood of replacement between each pair of amino acids, based on observed counts of substitutions in multiple alignments of evolutionarily-related proteins. The pairwise similarity scores can be used directly in most alignment algorithms as partial match scores (Sankoff and Kruskal, 1983).

One of the most commonly used tables by molecular biologists is MDM78 (Dayhoff *et al.*, 1978). We re-computed all pairwise alignments among the proteins in our dataset using

MDM78 for partial match scores. For fairness, we optimized the gap-weight parameters for this substitution table using a simple grid technique, as we did for our experiments, finding the best gap-open-penalty to be 1.2 and the best gap-extension-penalty to be 0.08. The average z-score when using the MDM78 substitution table was found to be 5.60, an increase of +0.38 over identities (indicated by the horizontal line in Figure 6.9). So our search method found a representation even better for homology modeling than using substitution tables (5.64 vs. 5.60).

6.4.5 Conclusion

From Experiment 2, we conclude that re-representing amino acid sequences in terms of properties can improve the ability of homology modeling to predict protein structures (as measured by z-scores). In contrast to Experiment 1, the success of this experiment comes from encoding the context-dependence of the relevance of various amino acid properties. The pattern of hydrophobicity among the six sequential neighbors at each site is an effective approximation of context. Using search to pick the best property for each context, a context-dependent representation was found by search that was better for homology modeling than amino acid identities. This representation was also better than using the MDM78 substitution table in alignments, demonstrating the significance of our approach by comparison to one of the most widely used compentitive methods for improving sequence alignments.

6.4.6 Analysis

To state the improvement made by this representation in terms more familiar to machine learning, we can convert z-scores into estimates of error rates. Using standard statistical tables (or a numerical approximation to the error function) z-scores can be translated into p-values, the probability that an alignment score from the background distribution (between unrelated proteins) would be greater than the observed score. This corresponds to the probability of making a mis-prediction because a protein in another fold class appear more similar to a query protein. By averaging the p-values for all pairs of proteins in the test set, the error rate for amino acid identities is found to be 27.4%, the error rate for MDM78 is 26.7%, and the error rate for the best context-dependent representation found by our search is 25.8% (these results are summarized in Table 6.8). As with z-scores, this analysis in terms of error rates confirms

Table 6.8: Summary of Results.

representation	*average z-score*	*delta z-score*	*avg. error*	*delta error*
identities	5.22	+0.00	27.4%	-0.0%
MDM78	5.60	+0.38	26.7%	-0.7%
best context-dependent	5.64	+0.42	25.8%	-1.6%

that our method found a representation (Table 6.7) that is better for homology modeling than either identities or substitution tables, by nearly 2% and 1%, respectively. Though our approach makes a seemingly small improvement in error rate, this could potentially enable identification of structures for hundreds of new sequences in our vast genomic databases.

Figure 6.10 shows the relationship between the change in z-score when using the best context-dependent representation and the original alignment score (using amino acid identities, approximately equal to percent identity (minus cost of gaps)) for each pair of sequences in the test set. There is little apparent correlation, though the improvement tapers off somewhat for highly homologous sequences because they have little room for improvement. The important thing from the perspective of molecular biology is that the new representation improves the significance for many low-homology alignments, since these are the least confident and are responsible for the most errors in modeling protein structures.

To get a feel for how this new context-dependent representation actually affects sequence alignments, we can look at both the new matches introduced into particular alignments, as well as any shifts in the location of gaps. To illustrate, we selected an alignment with one of the greatest increases in z-score: between the N-terminal half of the heavy chain of the Fab fragment of human IgG (2fb4) and the N-terminal half of the light chain of the Fab fragment of mouse IgA (2fbj). These are immunoglobulins with a beta-sandwich structure (two beta-sheets packed face-to-face). Figure 6.11 presents the alignment based on identities, the alignment based on our new representation, and an alignment constructed manually based on structural correspondences.

In this case, the new representation adds only three new matches. However, these new matches are enough to cause shifts in the location of gaps, resulting in a different alignment. The alignment score for identities is 0.153 ($\sim$15%), which has a z-score of 0.534 (assuming gap-open-penalty of 1.5 and gap-extension-penalty of 0.2, relative to a match score of 1). Using our context-dependent representation, the alignment score increases to 0.225 ($\sim$22%), and

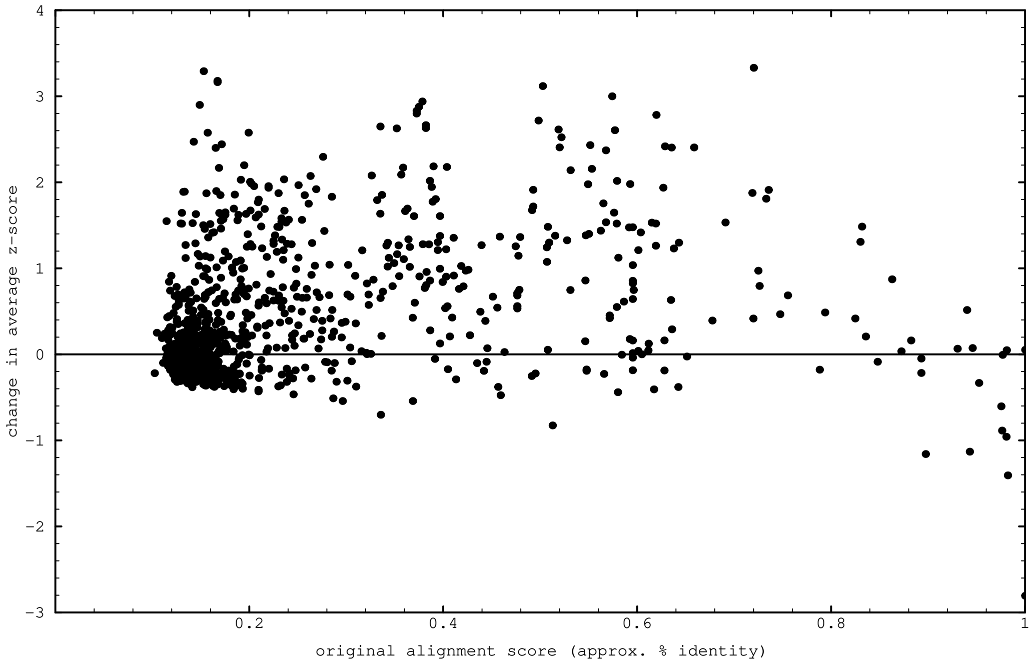

Figure 6.10: Dependence of change in z-score on percent homology (original alignment scores: percent identity minus gap costs) when using the improved representation from experiment 2. '0' on the delta-Z axis corresponds to the average z-score for identity matching: 5.22.

the z-score increases by over 3 standard deviations to 3.84. The alignment using our context-dependent representation seems slightly better than the identity alignment (closer to the structural alignment) in the first third of the protein, slightly worse in the middle, and just as bad in the last third. Formal measurement of the effect of such re-representations on sequence alignments remains for future work. However, it is clear that alternative representations will have to introduce many more new matches to improve the quality of alignments in addition to their significance.

6.4.7 Possible Extensions of this Experiment

There are several ways in which this experiment could have been done differently. In terms of search, we could have selected multiple properties into each context. After finding the best property for each context, as done in the current experiment, we could make a second pass to try merging together other subsets of residues. Perhaps properties that were found to be second-best in the first pass would the top candidates in the second pass. Perhaps a Monte Carlo approach could taken in which a random property is chosen for a random context, and

```
a. identity alignment

EVQLVQSGGGVVQPGRSLRLSCSSSGFIFSSYAMYWVRQAPGKGLEWVAIIWDDGSDQHY
|  | ||                |     ||   |              |
EIVLTQSPAITAASLGQKVTITCSASSSVSSLHWYQQKSGTSPKPWIYEISKL-------

ADSVKGRFTISRNDSKNTLFLQMDSLRPEDTGVYFCARDGGHGFCSSASCFGPDYWGQ
|  |  ||  |   |     |       ||   | |           |
ASGVPARFSGSG--SGTSYSLTINTMEAEDAAIYYCQQWTYPLITFGAGTKLELK---


b. alignment with context-dependent partitions

EVQLVQSGGGVVQP-GRSLRLSCSSSGFIFSSYAMYWVRQAPGKGLE-WVAIIWDDGSDQ
|  | ||        |  *   || |    |     |  |  |   * |   |    |
EIVLTQSPAITAASLGQKVTITCSASSSVSSLH---WYQQKSGTSPKPWIYEISKLASG-

HYADSVKGRFTISRNDSKNTLFLQMDSLRPEDTGVYFCARDGGHGFCSSASCFGPDYWGQ
     |  ||  |   |     |  *    ||   | |           |
-----VPARF--SGSGSGTSYSLTINTMEAEDAAIYYCQQWTYPLITFGAGTKLELK---


c. structural alignment

EVQLVQSGG-GVVQPGRS-LRLSCSSSGFIFSSYAMYWVRQAP--GKGLEWVAI
|  | ||        |         ||             |         |
EIVLTQSPAITAASLGQKVTITCSASSSV-----SSLHWYQQKSGTSPKPWIYE

IWDDGSDQHYADSVKGRFTISRNDSKNTLFLQMDSLRPEDT-GVYFCARDGGHGFCSSASCFGPDYWGQ
|         |  || ||  |         |       ||    |                 |
I------SKLASGVPARFSGSGSG--TSYSLTINTMEAEDAAIYYCQQWT----YPLITFGAGTKLELK
```

Figure 6.11: Effect of the context-dependent representation on the alignment of two immunoglobulins: 2fb4-H (top sequence) and 2fbj-L (bottom sequence). The top alignment was done with identities. Vertical bars indicate identity matches. The second alignment was done with our improved representation. The asterisks indicate matches based the context-dependent partitions, rather than identity. The third alignment was constructed manually from comparison of the structures.

then hill-climb for as many iterations as possible. More generally, we could try swapping amino acids around in these sets of partitions, as in the first experiment. However, computational complexity must be closely monitored; the search in experiment 2 already takes on the order of 10 hours on the CM5.

A more significant way to extend these experiments would be to define context differently. In our experiment, we defined context in terms of the pattern of hydrophobicity of the sequential neighbors $i-3 \ldots i+3$ to each site i. However, it is possible that other windows of amino acids could be used to define context, such as $i \pm 4$ or perhaps non-contiguous regions. Furthermore, other properties of these contextual residues in addition to hydrophobicity may be important, such as size or flexibility. In fact, a broader search might dynamically manipulate the definition of context itself, rather than just the partitions in each context, perhaps exploring the same list of interesting properties as for the partitions.

One interesting proposal would be to try merge some of the contexts together, forcing them to use the same partition in transformations. This would make sense if multiple contexts distinguished by our formalism are in fact placing similar constraints on the central residue. For example, suppose background knowledge suggested that contexts in which the residues at $i-2$ and $i+2$ are both hydrophobic are likely to be similar. This constraint would allow one partition to be used for 16 individual contexts, reducing the complexity search because they could be optimized together.

Another way to extend these experiments would be to add more knowledge about amino acid properties. Besides simply adding to the list of 18 properties we used, knowledge could be used to suggest possible correlations among the groups of amino acids. For example, if the small-polar group is found to improve z-scores when selected into a particular context, it would be especially reasonable to try the small-strong-polar group next.

Other knowledge about properties that would help the search in this experiment could come from studies of amino acid substitution patterns in certain secondary structures. For example, it has been observed that beta-branched residues are preferred in beta-sheets, and are found less frequently than expected in alpha-helices (Richardson and Richardson, 1989). Knowledge of the distributions of amino acids in these sub-structures could be used to estimate the contexts involved, establishing a direct relationship between context and property.

Chapter 7

Analysis and Evaluation

There are several ways to evaluate our framework. First, we look pragmatically at its performance in a difficult, real-world domain. The results of our experiments in protein structure prediction constitute empirical evidence for the utility of our framework. So we begin with an analysis of the role the framework played in generating these results. More generally, we can evaluate our framework by examining how well it solves the problems we set out in the introduction. We discuss the strengths and weaknesses of our framework for learning in real-world domains, in comparison to related methods in machine learning. Finally, any good scientific theory should have consequences that open up new directions for study. We conclude this chapter by proposing how the framework might be extended to evaluate domain knowledge, which would enable greater interaction between the expert and learning system.

7.1 Analysis of Role of Framework in Experiments

Our framework for change-of-representation played a central role in our experiments in protein structure prediction. The framework enabled us to achieve significant results in this difficult real-world domain. In our experiments, we increased the average z-score for homology modeling from 5.22 to 5.64, compared to the original representation in terms of amino acid identities. Since z-scores measure the significance of alignment scores, this improvement reflects increased sensitivity of sequence alignments for detecting structural similarities among proteins, and hence improves the ability of homology modeling to predict fold classes. To compare, we showed that our approach improved homology modeling even more than the most popular competitive method: using the MDM78 substitution table (Dayhoff *et al.*, 1978).

Furthermore, our improvement was achieved without resorting to tertiary structure data as an extra source of information, as is done in Eisenberg's 3D-1D profiling method (Gribskov *et al.*, 1987) and Lathrop's ARIADNE/ARIEL method (Lathrop *et al.*, 1993). Though no direct comparison was made to these systems, they are likely to improve the predictive ability of homology modeling beyond our approach, since knowledge of at least one of the structures of the two sequences being aligned can greatly inform the local matching process. However, in many sequence alignments, tertiary structures are not available for either of the proteins being compared, so these other systems would not be applicable. We endeavored to improve the quality of alignments more generally based only on information in the sequences themselves, in order to apply to a wider range of situations, even when structures are not known. However, it is also possible to imagine incorporating tertiary structure information into our re-representation scheme, if so desired, such as by using solvent accessibility as a context for re-naming amino acid identities by their properties.

The success of our approach stems from the power of re-representation. Amino acid sequences are a low-level representation for proteins because the identities of the amino acids in a sequence do not indicate much information about their role in the protein structure. By transforming these sequence elements into symbols encoding the physical and chemical properties of the amino acids, the local matching during sequence alignments can involve more intelligent decisions. The substitution-table method also tries to enhance local matching, but this approach necessarily collapses all the different dimensions of similarity into a single scalar value between each pair of amino acids. In contrast, we showed that our re-representation approach can be extended to select the dimension of similarity among multiple properties that is most relevant for matching at each site by taking context into account. The discovery of our most successful change-of-representation (end of experiment 2) was the result of cooperation between a human expert and an implementation of our framework. Based on background knowledge, the human suggested the form of the re-representation (i.e. context-dependent sequence transformations into strings over sets of property symbols), but the particular mapping (i.e. most relevant contexts for each property) was found empirically by the system by doing search.

One advantage of using our framework in these experiments is that it could be adapted to improving homology modeling, since the bias in this domain-specific algorithm is already known to be appropriate for protein structure prediction. Some change-of-representation techniques, such as FRINGE (Pagallo and Haussler, 1990), require the use of a specific kind of induction algorithm, which would probably result in an initial decrease in performance from the current established level. Furthermore, most known change-of-representation techniques are based on feature construction (Matheus, 1989), which assumes that examples can be expressed as feature-vectors of uniform length; but proteins are fundamentally variable in length. There are some methods known for learning sequences (Fu, 1982; Dietterich and Michalski, 1986; Rabiner, 1989), but these do not necessarily generalize sequences in a way useful for learning to recognize protein folds. Our approach, based on changing representations, is independent of the learning algorithm, which allows us to take advantage of the previous effort that has been expended on protein structure prediction by improving the best known method directly, rather than wasting human and computer effort on trying to catch up to its performance with another method.

Another advantage of using our framework in these experiments is that it enabled us to make use of domain experts' knowledge about amino acid properties. For very difficult tasks like protein structure prediction, we want to bring as much knowledge to bear on the problem as possible. Biophysicists have worked hard to understand the forces that are involved in determining protein structures, and have accumulated rules-of-thumb for various roles that individual amino acids might play. However, there are few systematic ways known for using this knowledge to improve homology modeling as a predictive method. Usually, homology modeling considers only the identities of amino acids in ordinary alignments of amino acid sequences. Amino acid properties are sometimes used in *post hoc* justifications of alignments to demonstrate conserved sites, though this does not improve the initial *prediction* of those structures. Substitution tables ignore this background knowledge altogether, instead resorting to statistical summaries of observed substitutions to define the similarities among the amino acids (Dayhoff *et al.*, 1978). Approaches such as ARIADNE/ARIEL (Lathrop *et al.*, 1993) and PIMA (Smith and Smith, 1990) can incorporate knowledge of amino acid properties into pattern-based representations of proteins. However, these systems require multiple alignments among a set of related sequences to determine the constraints on each site for generalization. In many cases, similar sequences may not be available for constructing multiple alignments, but our approach based on changing the representation of individual sequences can still be applied.

Our approach enabled us to take advantage of the available knowledge of amino acid properties by relating it to representation, which we found to be straightforward. In each experiment, the initial domain theory explained that homology modeling recognizes structural similarities between proteins based on the similarity between their sequences, and that sequences that reflect local structure should be best for identifying global structural relationships. By getting more specific about what is meant by 'local structure,' we were led to supply statements about how the amino acid properties might be good surrogates for local structure (in experiment 1), and about how sequential neighbors can influence local structure (in experiment 2). In each case, the existence of the various kinds of sequences for each protein was expressed using sequence transformation functions that mapped between them, which automatically generated alternative representations. This illustrates the natural connection between knowledge and representation on which our framework relies.

Our framework was able to handle uncertainty in this domain knowledge in an appropriate way. Despite the knowledge of many physical and chemical properties of amino acids that might play a role in proteins, biophysicists generally do not know which properties are most important for determining protein structure. The large number of potentially relevant properties for each amino acid overlap in complex ways, and the knowledge in this domain is not strong enough to decide which among these is most important. Previous methods in homology modeling that have used amino acid properties have usually ignored this uncertainty by picking a single, static set of property definitions that is thought to be a good approximation. For example, PIMA (Smith and Smith, 1990) uses a 4-level hierarchical classification of amino acids according to a few dominant properties. However, this approach trivializes many of the interesting relationships among the amino acids, and such simplified theories do not sufficiently explain the variety of substitution patterns that have been observed at different sites.

The search aspect of our framework enabled us to deal gracefully with this uncertainty in our experiments, essentially by letting the uncertain aspects of the knowledge generate a space of alternative representations. Our experiments focused on two particular issues of uncertainty in the knowledge: the best way to define the properties in terms of variations of amino acid subsets, and the most appropriate contexts in which each property should be used. Search was used to optimize over these uncertainties, dynamically finding the best parameters. Furthermore, the results were specifically tuned for the task of predicting protein structures, since the performance of homology modeling was used directly as the constraint for optimization. As a consequence, we were able to make significant use of the available background knowledge without forcing an over-commitment to a specific theory of amino acid properties. This demonstrates the effectiveness of our framework in exploiting the fine separation between what is known and what is unknown by domain experts.

In both experiments, once the initial domain knowledge had generated a space of alternative representations, additional knowledge was required to help focus the search by suggesting which alternatives were most reasonable to evaluate at each stage. The restrictions on search were important because the spaces were very large ($\sim 5.2 \times 10^{13}$ possible partitions of the 20 amino acids in experiment 1, and 18^{64} ways to use the 18 properties in 64 distinct contexts in experiment 2). Domain knowledge about which alternatives were expected to be better than others was easily found, and efficient search procedures were derived directly from these constraints. In experiment 1, we encoded the knowledge that the Dayhoff partition or a slight variation of it was expected to be useful for homology modeling. This statement resulted in a hill-climbing procedure for exploring a small region of the space around the Dayhoff partition by randomly swapping amino acids between classes. In experiment 2, we encoded the belief that the utility of transforming amino acids according a particular property was probably independent between contexts but correlated with the utility of other properties in the same context. This led to another specialized search procedure in which the contexts were optimized one at a time, greatly reducing the size of the sub-space explored (resulting in the evaluation of only 1152 representations). These observations support the feasibility of generating efficient procedures for searching for improved representations as a direct consequence of background knowledge about a domain.

7.2 General Analysis of Framework

At a higher level of evaluation, our search-based framework for change-of-representation solves the two problems for machine learning in real-world domains that we posed in the introduction. First, our framework handles the complexity of change-of-representation by using domain knowledge to help decide which alternatives to explore. The space of alternative representations is often very large, and any method for change-of-representation needs a way of guiding the search to make the discovery of improved representations efficient. Our framework relies explicitly on domain knowledge to make these decisions, allowing it to make more intelligent choices that are specific to each domain. We compare our framework in detail to other methods for change-of-representation, which have generally been less knowledge-intensive.

Our framework simultaneously solves the second common problem for machine learning in real-world domains by providing a systematic way of incorporating domain knowledge. Typically, background knowledge is available in real-world domains, but it may not easily fit into any of the known roles, such as explanation or analogy, or they might not be flexible enough to deal with uncertainty in the knowledge. Our framework incorporates knowledge by adapting it to suggest alternative representations, which is a guideline for a domain expert to express his or her knowledge in a way that is relevant. It may not always be obvious how to relate knowledge to representation, though there is often an intuitive connection in real-world domains. However, once translated into this form, our framework gives a general algorithm for using it to improve the accuracy of a learner by searching for higher-level representations. We discuss the generality and flexibility of this role for knowledge, and we conclude this section with a discussion of the limitations of our approach.

7.2.1 Comparison to Other Change-of-Representation Systems

7.2.1.1 Feature Construction

The approach to change-of-representation in machine learning that is closest to our framework is feature construction (Matheus, 1989). Feature construction refers to the generation of new variables by applying operators, often mathematical or logical, to attributes in the original feature vector. Values for constructed features are computed for each example and appended to its description. Some example systems that construct features are FRINGE (Pagallo and Haussler, 1990), STAGGER (Schlimmer, 1987), CITRE (Matheus and Rendell, 1989), and LFC (Ragavan and Rendell, 1993). Matheus (1989) proposes a general framework that captures the similarities among such feature construction systems in terms of four phases: 1) detecting the need to new construct features, 2) selecting an operator and features as arguments to combine, 3) evaluating and choosing the best candidate, and 4) generalizing this feature.

Like our framework for change-of-representation, Matheus' framework for feature-construction shows that it can also be thought of as searching for representations that improve the accuracy of an underlying learning. However, a major difference is that the search space in feature-construction consists of constructed features, whereas in our framework, the search space consists of whole representations. Feature construction takes advantage of the natural decomposition of feature-vector-based domains to simplify the task of optimizing their representations. However, this approach consequently restricts the applicability of feature construction to domains where example can be described by fixed-length feature vectors. In domains where examples take on variable-length descriptions, such as sequences or trees, it is not clear how to apply feature construction methods. An advantage of our framework is that it does not depend on any syntactic aspect of the representation, and hence can be used to change representations in a wider class of domains.

Because feature construction searches through a different space, it uses a fundamentally different approach to evaluating nodes. Many feature construction systems use a feature evaluation method to compare features discovered during search. For example, STAGGER evaluates the utility of features by computing a function of the number of positive and negative examples covered by each construction. Other approaches to feature evaluation include blurring (Rendell and Ragavan, 1993) and contextual merit (Hong, 1994). In comparison to our framework, evaluating independent features may be more efficient, since bad ones can be ruled out once, rather than tested with many different representations. However, features are not always independent, and utility might depend on the other features in the vector (e.g. "group effectiveness;" Seshu *et al.*, 1988), making it difficult to define a meaningful utility for a feature out of the context of a specific representation.

Feature construction systems typically use a specialized method to decide which features to construct, corresponding to the strategy component in our framework. For example, FRINGE looks for patterns of variables that recur at the leaves of a decision tree (Pagallo and Haussler, 1990). BACON looks for mathematical combinations of numeric variables that are nearly constant (Langley *et al.*, 1983). LFC uses a geometric constraint to rule out combinations unlikely to decrease entropy, and then uses lookahead to dynamically explore a limited amount of feature space to decide which features to construct (Ragavan and Rendell, 1993). However, these approaches tend to de-emphasize the role of knowledge. Our framework is intentionally designed to replace complex techniques for selecting operators, operands, etc. with domain knowledge, making feature construction more *knowledge-intensive*. A system that illustrates how knowledge may be used in feature construction is MIRO (Drastal *et al.*, 1989), which restricts constructions to those that can be deduced from a background Horn-clause theory. Donoho and Rendell (1996) discuss other roles that knowledge can play in feature construction.

7.2.1.2 Constructive Induction and Dynamic Bias

Another approach to change-of-representation, constructive induction (Michalski, 1983), is more general than feature construction because it is based on a logical formulation of learning. Examples are described using predicates, and background knowledge is encoded as statements in annotated predicate calculus (APC). Induction is seen as a two step process: plausible generalizations of a set of examples are inferred from a domain theory, and then a preference criterion (e.g. lexicographic-evaluation function, minimum description-length principle, etc.) orders these hypotheses and selects the best one. Change-of-representation can be accomplished by adding more rules in APC that create new terms for the description language in which the hypotheses can be expressed. So this approach incorporates domain knowledge in a direct way that is very similar to how our strategy layer suggests alternative representations. However, in contrast to our framework, the search takes place entirely within hypothesis space, and there is no explicit evaluation of individual representations.

Our framework can also be compared to systems that use a dynamic or variable bias, such as the Variable-Bias Management System (VBMS; Rendell, 1987). This view is based on treating the external representation of examples as an aspect of bias, which is anything that affects generalization besides the examples themselves (Mitchell, 1980). For example, STABB shifts the symbolic language for describing mathematical functions to prevent version-space collapse in LEX, a system for learning how to do integration (Utgoff, 1986). Some systems treat all external factors in a learner as parameters, and then attempt to optimize predictive accuracy over this space (e.g. AIMS; Tcheng *et al.*, 1989). NTC shows how this can be applied to feature construction (Seshu *et al.*, 1988). Chrisman (1989) presents a criterion based on the PAC-learning model for deciding when to switch biases.

Dynamic bias methods such as these have tended to emphasize automation, and typically do not provide any roles for domain knowledge. An exception is Cohen (1990), who attempts to integrate knowledge with change-of-representation by inferring alternative representations with an "appropriate" bias for learning, though this is oriented toward exact learning models and does not involve any search. However, Gordon and DesJardins (1995) present a general framework for dynamic bias that is based on search and can incorporate explicit domain knowledge in a way similar to our framework. The model is based on multiple layers of search, each of which gives a space of ways for searching at the level below. The bottom layer is a hypothesis space, as is typical for models of learning (Mitchell, 1982), and the top level must ground out in some absolute search methodology dependent on background knowledge. In our framework, the middle layer is specifically chosen to be a space of representations to give more structure to the relationship between knowledge and learning.

7.2.1.3 Miscellaneous Systems

Several other machine learning systems can be described as using change-of-representation. For example, Dietterich and Michalski (1986) propose several operators to create new representations in SPARC/E, a system for learning how to play a card game by generalizing game sequences. The re-representation operators, which are typically parameterized, include augmenting sequence elements with derived attributes, breaking the sequence into non-overlapping segments, and selecting sub-sequences. Different parameters for these transformations were tried until an adequate model of the target concept could be learned. Hirsch and Japkowicz (1994) propose a domain-specific method for change-of-representation to improve the recognition of promoters in DNA sequences. A domain expert suggested that the occurrence of a particular subsequence was probably relevant because of how it twisted the strand to interact with DNA-binding proteins. Hirsch and Japkowicz generalized this knowledge to try extracting other features with a similar syntactic structure. Even MARS (Friedman, 1991) can be considered to do change-of-representation in statistical context. MARS is based on a parametric method for fitting spline curves to multi-dimensional functions. However, MARS extends this approach by dynamically multiplying variables together to get new dimensions which are smoother.

Our framework captures change-of-representation at a deeper level that unifies all these seemingly disparate systems and exposes their underlying commonalities. Each of these systems can be described as searching through a space of alternative representations to improve the performance of a more basic induction algorithm. Our definition of representation is sufficiently abstract that it does not matter whether the examples in a domain are represented as fixed-length feature vectors (e.g. CITRE; Matheus and Rendell, 1989) or variable-length descriptions such as sequences (e.g. SPARC/E; Dietterich and Michalski, 1986), nor whether data types of attributes are Boolean (e.g. FRINGE; Pagallo and Haussler, 1990) or numeric (e.g. MARS; Friedman, 1991). Similarly, all these systems generally have the goal of improving the performance of the learner, though there are many variations on the particular evaluation mechanism from cross-validated accuracy (e.g. NTC; Seshu *et al.*, 1988) to version-space collapse (e.g. STABB; (Utgoff, 1986)) to variance minimization (e.g. MARS; Friedman, 1991) to PAC-criteria (Chrisman, 1989). Finally, to the degree that domain knowledge is involved, it can be understood as helping to choose alternatives to explore. Examples include the restriction of constructions in MIRO (Drastal *et al.*, 1989), the choice of sequence re-representation operators in SPARC/E (Dietterich and Michalski, 1986), the APC rules in Michalski's (1983) framework, and the syntactic class of features to extract from DNA (Hirsch and Japkowicz, 1994). Our framework reduces the surface-level dissimilarities among all these approaches, and shows how they all instantiate the same fundamental principle of searching for representations that improve inductive performance.

7.2.2 Role for Domain Knowledge in Machine Learning

An important consequence of our framework is that it exposes a systematic way of incorporating domain knowledge into machine learning. In our framework, knowledge is used in the strategy layer to suggest alternative representations. We described various ways that knowledge can guide the search through the space of alternatives, such as delimiting the space, or acting as constraints or preferences. More generally, knowledge can be seen to act as a *heuristic* This role for knowledge takes advantage of a traditional concept in AI. Hard problems are often described in terms of large search spaces (Newell and Simon, 1972). This applies especially to machine learning where the space of alternative representations is often huge. As in many other problems, knowledge can help by steering the search more efficiently toward solutions. Hence the role we have found is actually an instantiation of a principle that has been known to the AI community for a long time. It is a well-understood way of enabling a human to help improve the performance of a program by giving it expert advice.

The way knowledge is incorporated in our framework is both general and flexible, which are two important criteria for any role for knowledge. First, this role is general because it does not depend on the inductive algorithm. The inductive algorithm is treated as a module to be plugged into the evaluation component with a simple interface. The only assumption is that the algorithm can be given sets of examples to train and test itself, and will return some performance score.[1] Thus the framework can be applied to improving an arbitrary algorithm, including those based on variable-length representations (unlike feature construction, which is restricted to domains where examples are represented by fixed-length vectors). This is especially important in real-world domains, where a domain-specific algorithm may be preferred. Our framework allows an expert to choose whatever inductive algorithm is believed to have an appropriate bias for the domain.

Another reason that the role for knowledge in our framework is general is that a great deal of knowledge can be related in a natural way to representation. The semantic interpretation of attributes as subsets of objects with a given value makes them equivalent to logical concepts, which are relations over the objects in model theory (Michalski, 1983). The primitives in symbolic descriptions, along with their syntactic relationships, reflect semantic relationships among domain objects via the denotation function. Thus other concepts that cover different subsets of objects can become new syntactic elements, which is most clearly illustrated by encoding intermediate concepts as new attributes (Fu and Buchanan, 1985). These might help learning if they share boundaries with the target concept, because they give the learner hints for where to extend generalizations (Utgoff, 1986). Other examples where knowledge can lead to new subsets of data that might be closer to the target concept include discretization of numeric variables and grouping of nominal variables. Knowledge of interactions among variables, such as correlations or functional dependencies, perhaps based on relationships like spatial proximity, can also lead to changes in representation by constructing combination variables that draw examples closer together in higher dimensions (Davies and Russell, 1987; Ragavan and Rendell,

[1] Any parameters the induction algorithm depends on must be determined in advance or adjusted automatically by a wrapper algorithm that makes it invisible to the evaluation component.

1991). Deeper explanations (e.g. causal models) may be linked to the base features using standard logic, as illustrated in our earlier example about the relationship between infections and raised body temperatures.

The role for knowledge in our framework is also flexible in the sense that it can tolerate uncertainty. Knowledge in real-world domains can often be incomplete, incorrect, inconsistent, etc. (Wilkins, 1990). Some roles for knowledge can be sensitive to this and will either fail to learn or learn incorrect generalizations in such situations (e.g. EBL; DeJong and Mooney, 1986). Because of the search basis of our framework, it is inherently tolerant of uncertainty. An incorrect piece of knowledge in the strategy component might steer the search into unproductive parts of the space, causing the system to explore many alternative representations which do not turn out to improve the accuracy of the learner. However, assuming the search will always keeps track of the best representation found so far, this mis-guidance merely wastes time, which is a more appropriate penalty for weak domain knowledge. The search basis of our framework makes clear the tradeoff between the strength of the knowledge and the efficiency of change-of-representation (Newell and Simon, 1972).

7.2.3 Limitations of the Approach

Our framework cannot be used to improve induction in arbitrary domains. In this section we discuss some criteria, which, if not met in a given domain, could prevent successful application of the approach. One premise of the framework is that the domain is initially difficult for learning, so that current induction algorithms get only low accuracy. Furthermore, this difficulty should be due to a low-level representation, rather than other causes such as noise (Quinlan, 1986) or intrinsic inaccuracy (Rendell and Seshu, 1990). Another premise is that domain knowledge is available which can be used to help change representations. Our framework is semi-automated, requiring knowledge. It will not work in artificial domains lacking significant background knowledge. However, low-level representations and background knowledge are common characteristics of real-world domains.

More interesting limitations stem from qualities of the knowledge that are required for the framework to work. First, it must be possible to relate the knowledge to representation. While we claim that there is often a fairly natural connection in many real-world domains, our framework does not automatically transform arbitrary domain knowledge into changes-of-representations, nor has this research focused on cataloging roles for knowledge to guide the domain expert (Donoho and Rendell, 1996). The framework merely provides a general guideline for expressing knowledge (i.e. as a partial order among alternatives that are expected to improve learning), and it remains the expert's responsibility to find a way of adapting knowledge of the domain for this purpose. Furthermore, getting specific about changes-of-representation might necessitate reasoning about syntactic manipulations (e.g. list operations), which can be tedious and is often at lower level of detail than most experts want to express.

Another potential limitation is that the expert might need to understand how the bias of the selected learning algorithm. For an expert to make suggestions of some representations that should be better than others, he or she must be able to understand how generalizations are made in order to predict the effect of various changes-of-representation on the accuracy of learning. However, this may not be as difficult as it appears. The biases of many common learning algorithms are often easy to comprehend, in the sense that learning is usually easier with features that have high similarity to the target concept. Even with domain-specific algorithms, experts often have an intuition about which syntactic components of example descriptions are getting generalized during learning, and the obvious goal is to map domain objects into descriptions that assign these components in such a way that they are correlated in some way with the target concept.

Finally, the domain knowledge must be relatively strong. The role of the knowledge in our framework is to guide the search for alternative representations. If the knowledge is not strong enough, then the search will be inefficient and the framework will not be effective. Although some uncertainty is allowed, it is insufficient to place arbitrary constraints on representations, which can result in an intractable search. The expert must supply constructive knowledge which makes explicit suggestions for search. The expert is left with the responsibility of controlling search complexity, and this can be awkward in our framework because it can be done only by manipulating the domain theory.

7.3 Future Work: Evaluating Knowledge to Support Interactive Learning

An important consequence of our framework is that it suggests a way of evaluating knowledge. In this section we show how this idea might be used to extend our framework to be more interactive. The framework establishes a connection between knowledge and search in the form of suggestions for alternative representations. By exploiting this link in reverse, we can use the results of search to critique the knowledge. Intuitively, if the knowledge suggests alternative representations that do not turn out to improve the accuracy of the learner, then it can be assumed that the knowledge is incorrect. The underlying assumption is that the correctness of the knowledge can be equated with its ability to suggest improved representations for learning, which is a form of *utility principle*.

In the analysis of our first experiment, we described a method for summarizing search results that illustrates this potential for evaluation of knowledge in a domain-specific way. We extracted patterns of associations between pairs of residues that were found in the same class in partitions that had been improved by search. We interpreted these patterns as indications of the correctness of the groups of amino acids Dayhoff used to define various properties. This domain-specific implementation yielded several important high-level results about amino acid properties, such as confirming the membership of residues like methionine and histidine in certain groups, and revising the definition of the small group by removing glycine, cysteine, glutamine, and proline. This is made possible by the direct role of the knowledge in the search: the knowledge is responsible for the representations that are found.

While the customized method used in our experiments was specific for the protein structure prediction domain, we can propose a more general implementation for evaluating knowledge, based on the formal connection between knowledge and representations made by our framework. Critiques of the knowledge can originate from observations of inconsistencies between the expected order for the accuracies of alternative representations and the actual order of their accuracies when evaluated. Recall that knowledge in our framework is applied to generate a partial order $>_j$ among alternative representations, given the current state j of the search. The semantics of $r_i >_j r_k$ is that, based on the previous representations evaluated up to iteration j of the search, the accuracy of the learner when examples are transformed into representation

r_i is expected to be greater than that for representation r_k. Suppose that, later in the search, representations r_i and r_k are evaluated, and are found to have accuracies a_i and a_k such that $a_i < a_k$. This indicates a problem with the theory, since the knowledge generated a partial order that was inconsistent with subsequent observations.

Given this ability to critique the theory as a whole, a more in-depth evaluation of knowledge would attempt to localize the parts of the domain theory that are incorrect, which is often called *credit assignment* (Dietterich *et al.*, 1982). While the abstract framework does not make any assumptions about the form of the domain theory, CORIL, our implementation in CLP(R), assumes that the domain knowledge is encoded in the form of a Horn-clause theory. This format for the domain theory is convenient for evaluation because the individual clauses can be treated as independent pieces of knowledge. The role of each Horn clause in any given decision can be determined by examining the proof structure supporting that decision. Specifically, for each pair of evaluated representations r_i and r_j for which the theory has an expectation about the relative order of their accuracies, the proof tree for $order(r_i, r_j)$ or $order(r_j, r_i)$ would be generated. This tree reveals the dependence of erroneous conclusions on individual parts of the theory. More specific localization of a fault could be achieved by accumulating statistics on the involvement of each rule over multiple successes and failures, as is done in SEEK2 (Ginsberg *et al.*, 1988).

This method for evaluating knowledge could be used to support a semi-automated version of *theory revision* (Pazzani, 1988; Towell *et al.*, 1990; Mooney and Ourston, 1994). Theory revision generally involves three steps: 1) detecting a fault in a theory, 2) determining which parts of the theory are responsible, and 3) suggesting repairs. The utility principle based on our framework solves the first problem, and the proposed credit-assignment method would solve the second problem. However, a fully-automated implementation of theory revision would require a technique for generating repairs, which would require making assumptions about the class of errors in which the fault is expected to be (Wilkins, 1990). Instead, the feedback produced by our framework could also be given directly to the domain expert, who is in a better position to understand how to repair the theory (Davis, 1982). A system based on our framework, could return critiques of individual rules in the theory, based on how useful they were during the search. It would then be the responsibility of the domain expert to decide how to update the theory. For example, the expert might know which rules are subject to revision because they are uncertain, and what syntactic aspects of those rules are involved in the uncertainty (e.g. arguments, antecedents, etc.).

Supplying this kind of feedback to the expert can lead to a more interactive model of learning. If the expert can revise the theory based on the system's critique of the knowledge, then the expert might change the input theory and re-start the search. This cycle can be iterated until the expert and learning system converge together on an adequate theory of the domain. This is particularly useful in real-world domains where the knowledge is uncertain, which is common if an understanding of the domain is only just being developed. The expert can then use such a system to test hypotheses interactively. For example, the expert might postulate a new rule, append it to the base domain theory (knowledge that is believed to be fairly stable), run the search through representation space, and examine the results to see if improved representations were discovered (specifically, see if the expected order of accuracies for alternative representations is predicted correctly more often). The ability to test hypotheses and interactively refine a domain theory can eventually lead to higher-level results by improving the expert's overall understanding of the domain, as demonstrated in our experiments.

Such interaction between the domain expert and the learning system might be crucial to the success of machine learning in real-world domains. Learning in difficult domains requires the unique resources of both a human and a computer. A computer (with an induction algorithm) can scan through huge amounts of data to look for patterns, and a human (i.e. an expert in the domain) has background knowledge that might be relevant to the problem. However, each of these participants needs some assistance. The computer needs help finding higher-level representations and knowing what patterns to look for, and the human needs a way of evaluating knowledge and testing hypotheses. Our framework brings these two needs together, allowing the human's expertise to facilitate the learner, and allowing the results of the learner to be translated into critiques of the knowledge. Thus the interactive model of learning supported by our framework makes the best use of available resources, and, as a result, can potentially lead to both an increase in the accuracy of the learner, and a deeper understanding of the domain by the expert.

Chapter 8

Conclusion

This thesis has focused on two pragmatic issues of learning in real-world domains which are inadequately addressed by current machine learning research. First, examples in real-world domains are often given in a low-level representation, which means that none of the initial attributes is strongly related to the target concept by itself. Low-level representations can cause a variety of phenomena that interfere with common biases in learning algorithms, and can thus prevent a broad class of algorithms from achieving high accuracy. So effective learning in real-world domains often requires finding higher-level representations. There are several automated methods for changing representations, such as feature construction and constructive induction. However, in general, the space of possible representations is very large, and each such method must have a way of dealing with this complexity. This often requires making implicit assumptions about which representations to consider. We need a way of making more intelligent choices for change-of-representation that can be tailored for each domain.

Another common characteristic of real-world domains is that some background knowledge is almost always available. Intuitively, it would seem that this extra resource (in addition to the examples) should provide some leverage for increasing predictive accuracy, and there may be great incentive to do so. However, most learning algorithms are designed to work only with databases. There are several roles known for knowledge to play in learning, such as explanation or declarative bias. However, these roles for knowledge are often restrictive, in that they require the use of a specific induction algorithm, or are inflexible, in that they might decrease accuracy if the knowledge is incorrect or incomplete. Thus we also need more general and flexible ways of incorporating domain knowledge into learning.

In this thesis, we have proposed a general framework for change-of-representation that integrates solutions to these two problems, based on using knowledge to help search through the space of alternative representations. We assume that some induction algorithm has already been selected, and it is the goal of the framework to improve the accuracy of this algorithm, which is held constant, by changing representations. We define representations, which become nodes in the seach space, as mappings of domain objects into arbitrary symbolic descriptions, which is more general than just a "list of features."

Our framework has three components. An evaluation component compares alternative representations by applying them to a set of examples, running the induction algorithm on the transformed examples, and estimating the effect of this change on accuracy by cross-validation. To determine which representations (nodes in the space) should be searched, a strategy component is invoked. The strategy component incorporates explicit domain knowledge in the form of suggestions for alternative representations. We provided a formal semantics for this knowledge in terms of a partial order among alternatives, as a function of the current search state. Finally, a central search component mediates between the strategy and evaluation components, iteratively selecting a suggested representation (i.e. "maximal" in the partial order), and evaluating it to monitor the progress of the search through the space.

This framework simultaneously solves the two problems mentioned above. First, domain knowledge acts as a heuristic to facilitate the search for alternative representations. This reduces the complexity of change-of-representation by taking advantage of special properties of the domain to steer the search more efficiently toward improved representations. Importantly, our framework is not restricted to feature-vector-based domains, and can be applied to domains where examples are represented in variable-length descriptions such as sequences, for example. Our approach is more knowledge-intensive than most feature construction methods, though as a tradeoff it is only semi-automated, depending on the assistance of a human. Pragmatically, an implementation of this framework would facilitate the interaction between a human expert and a learning program in which the expert would hypothesize and prioritize new representations, and the system would empirically evaluate them to identify the best one for learning.

The framework also provides a systematic way of incorporating domain knowledge into learning. Specifically, knowledge is used in our framework to suggest alternative representations for search. This role involves both defining what kinds of re-representation might improve learning, as well as giving expectations about which particular alternatives might be best to try, which are guidelines the domain expert can use for expressing his or her knowledge in a way that is relevant. It may not always be obvious how to relate knowledge to representation, and our framework does not provide any direct assistance to the expert in this matter. However, in many real-world domains, it is often intuitive how to adapt knowledge for this purpose because of the natural connection between knowledge and representation (e.g. from an understanding of the meaning of the features and their relationships).

This role for knowledge is general in that it can be used to improve an arbitrary learning algorithm, since the algorithm is only treated as independent module by the evaluation component. This is important in real-world domains because it allows the use of whatever algorithm is believed to have an appropriate bias for the domain. Furthermore, this role is tolerant of uncertainty. Incorrect knowledge might steer the search into unproductive regions of the space, but this only wastes time, instead of decreasing accuracy. So suggesting alternative representations is a flexible way for an expert to give advice to a learning program in a way that can be used to improve its performance.

We applied our framework to an important real-world problem: protein structure prediction. Proteins are complex molecules in cells, composed of linear chains of amino acids. The best method to date for predicting the 3D structure of a protein from its amino acid sequence is homology modeling, which is based on sequence alignments with a protein database. A major limitation of homology modeling is that it can fail due to the hidden-homology problem, which is when proteins with similar structures do not appear to have high sequence similarity.

In beginning to apply our framework to this problem, we showed how homology modeling could be treated as a learning algorithm (i.e. nearest-neighbor). For domain knowledge, we noted that biophysicists believe the physical and chemical properties of amino acids are relevant to protein structure. In working out the details of this knowledge (via increasingly specific definitions), it automatically revealed ways to change the representation of amino acid sequences, for example by replacing the identities of the amino acids at each site with their properties.

Additional knowledge suggested that definitions of amino acid properties similar to those given by Dayhoff et al. (1972) might be useful. These constraints led to an efficient sequence of suggestions for search that was equivalent to a form of hill-climbing. The Dayhoff transformation of amino acids turned out to be worse for homology modeling than the original identities (accuracy: identities=77.7%, dayhoff=56.4%). However, our search process was able to discover significant improvements of the Dayhoff partition (average=74.8%). While on average, the improved representations did not exceed the performance with identities either, this experiment demonstrated that by incrementally perturbing and evaluating partitions of amino acids, some representations (e.g. Dayhoff) could be improved.

In a second experiment, we encoded more complex knowledge about the context-dependence of the relevance of amino acid properties. Knowledge of biophysics suggests that the pattern of hydrophobicity among the three N-terminal and three C-terminal neighbors might be a good indicator of context, which was incoporated into representation by selecting which properties would be used for transforming each site. Additional knowledge gave expectations about which of these context-dependent representations was most likely to improve learning, based on an assumption of independence among the contexts, and a set of interesting amino acid properties to consider. This knowledge also led to an efficient search procedure, which was able to eventually discover a representation that was even better for homology modeling than the best known competitive method of using substitution tables (average z-scores: identities=5.22, MDM78=5.60, search=5.64).

These significant results in a difficult, real-world domain constitute empirical evidence for the utility of our framework. Protein structure prediction is a difficult problem, particularly because sequences of amino acids identities are a low-level representation for proteins. Additional background knowledge, i.e. about amino acid properties, was available that was believed to be relevant. This knowledge was easily related to representation in the course of specifying what it means for various kinds of sequences (e.g. of local structure, roles, constraints, etc.) to reflect the global structure of a protein. However, the knowledge was uncertain, in that the best definitions and contexts for various properties were unknown. This illustrates the importance of the notion of a search space for allowing uncertain knowledge to be utilized without forcing a commitment to every detail, or trivializing the knowledge by approximation. Finally, the framework requires the knowledge to be strong enough to give explicit guidance for search, but expectations about which alternative representations should be best for homology modeling were easily derived from domain knowledge, resulting in efficient searches.

We discussed an important consequence of the framework: its ability to support increased interaction between a human expert and a learning program. In one direction, our framework uses representation as an intermediate language for a domain expert to give advice to the learner. We also suggested how the formal connection between knowledge and representation in our framework might be exploited in the other direction to evaluate knowledge based on the results of search. Critiques of the input knowledge could be given to the user as feedback, potentially enabling the user to test hypotheses and refine the domain theory. This kind of interaction could be crucial to successful learning in real-world domains where the user is currently evolving his or her understanding of the domain.

We illustrated this ability to evaluate knowledge in the analysis of our first experiment. By analyzing good partitions of amino acids found during search, we confirmed the membership of several ambiguous amino acids (e.g. histidine and methionine) in certain groups (positive and hydrophobic, respectively). We also found that the group of small residues should be reduced to six core amino acids (alanine, aspartate, glutamate, asparagine, serine, and threonine). Of the amino acids split out of the small group, the apparent uniqueness of glutamine was most surprising, suggesting that its role in protein structure needs to be re-examined. These high-level results are as important for biology as the discovery of specific representations that improve the performance of homology modeling.

Our approach is extensible to other real-world domains. The criteria for applying our framework are: 1) the initial representation must be low-level, 2) domain knowledge must be available, and 3) the knowledge must be made relevant to representation. We described an implementation based on CLP(R) called CORIL which could be used in other domains where knowledge can be expressed in the form of Horn-clause theories. However, CORIL is not the only way to implement the framework, and other implementations may be more appropriate in other domains. The key aspect of the framework is that knowledge is adapted to imply a partial order among alternative representations, based on which ones are expected to produce the greatest increases in accuracy. These suggestions can be used to guide a search for representations that are empirically observed to improve the accuracy of an underlying inductive algorithm.

References

Abu-Mostafa, Y. S. 1994. Learning from hints. *Journal of Complexity* 10:165–178.

Aha, D.W.; Kibler, D.; and Albert, M.K. 1991. Instance-based learning algorithms. *Machine Learning* 6:37–66.

Alberts, B.; Bray, D.; Lewis, J.; Raff, M.; Roberts, K.; and Watson, J.D. 1983. *Molecular Biology of the Cell.* Garland Publishing: New York.

Altschul, S.F.; Gish, W.; Miller, W.; Myers, E.W.; and Lipman, D.J. 1990. Basic local alignment search tool. *Journal of Molecular Biology* 215:403–410.

Amarel, S. 1970. On the representation of problems and goal-directed procedures for computers. In Banerji, R.B. and Mesarovic, M., editors 1970, *Theoretical Approaches to Non-numerical Problem Solving.* Springer-Verlag: Heidelberg, Germany.

Anderberg, M.R. 1973. *Cluster analysis for applications.* Academic Press: New York.

Baldwin, R.L. 1989. How does protein folding get started? *Theoretical Issues in Biological Sciences* 14:291–294.

Bax, A. 1989. Two-dimensional nmr and protein structure. *Annual Review of Biochemistry* 58:223–256.

Bergadano, F. and Giordana, A. 1990. Guiding induction with domain theories. In Kodratoff, Y. and Michalski, R., editors 1990, *Machine Learning: An Artificial Intelligence Approach, Volume III.* Morgan Kaufmann: San Mateo, CA.

Bernstein, F.; Koetzle, T.; Williams, G.; Meyer, E.; Brice, M.; Rodgers, J.; Kennard, O.; Shimanouchi, T.; and Tasumi, M. 1977. The protein data bank: A computer-based archival file for macromolecular structures. *Journal of Molecular Biology* 112:535–542.

Blundell, T.L.; Sibanda, B.L.; Sternberg, M.J.E.; and Thornton, J.M. 1987. Knowledge-based prediction of protein structures and the design of novel molecules. *Nature* 326:347–352.

Bohr, H.; Bohr, J.; Brunak, S.; Cotterill, R.M.; Lautrup, B.; and Petersen, S.B. 1990. A novel approach to prediction of the three-dimensional structures of protein backbones by neural networks. *FEBS Letters* 261:43–46.

Bowie, J.U. and Eisenberg, D. 1993. Inverted protein structure prediction. *Current Opinions in Structural Biology* 3:437–444.

Bowie, J.U.; Luthy, R.; and Eisenberg, D. 1991. A method to identify protein sequences that fold into a known three-dimensional structure. *Science* 253:164–170.

Breiman, L.; Friedman, J.H.; Olshen, R.A.; and Stone, C.J. 1984. *Classification and Regression Trees.* Wadsworth: Belmont, CA.

Brown, M.; Hughey, R.; Krogh, A.; Mian, I.S.; Sjolander, K.; and Haussler, D. 1993. Using dirichlet mixture priors to derive hidden markov models for protein families. In *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology.* 47–55.

Buchanan, B.; Mitchell, T.M.; Smith, R.G.; and Johnson, C.R. 1978. Models of learning system. In Belzer, J., editor 1978, *Encyclopedia of computer science and technology*, volume 11. Marcel Dekker: New York.

Cheeseman, P.; Kelly, J.; Self, M.; Stutz, J.; Taylor, W.; and Freeman, D. 1988. Autoclass: A baysian classification system. In *Proceedings of the Fifth International Workshop on Machine Learning.* 54–64.

Chothia, C. and Lesk, A.M. 1986. The relation between divergence of sequence and structure in proteins. *EMBO Journal* 5:823–826.

Chothia, C. 1988. The fourteenth barrel rolls out. *Nature* 333:598–599.

Chothia, C. 1992. One thousand families for the molecular biologist. *Nature* 357:543–544.

Chou, P.Y. and Fasman, G.D. 1974. Prediction of protein conformation. *Biochemistry* 13:222–244.

Chrisman, L. 1989. Evaluating bias during pac-learning. In *Proceedings of the Sixth International Workshop on Machine Learning.* 469–471.

Cohen, W.W. 1990. An analysis of representation shift in concept learning. In *Proceedings of the Seventh International Conference on Machine Learning.* 104–112.

Cooper, A. 1976. Thermodynamic fluctutations in protein molecules. *Proceedings of the National Academy of Sciences, USA* 73:2740–2741.

Cornette, J.L.; Cease, K.B.; Margalit, H.; Spouge, J.L.; Berzofsky, J.A.; and DeLisi, C. 1987. Hydrophobic scales and computational techniques for detecting amphipathic structures in proteins. *Journal of Molecular Biology* 195:659–685.

Davies, T.R. and Russell, S.J. 1987. A logical approach to reasoning by analogy. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence.* 264–270.

Davis, R. 1982. Application of meta-level knowledge in the construction, maintenance and use of large knowledge bases. In Davis, R. and Lenat, D.B., editors 1982, *Knowledg-Based Systems in Artificial Intelligence.* McGraw-Hill: New York. 229–490.

Davison, M.L. 1983. *Multidimensional Scaling.* Wiley: New York.

Dayhoff, M.; Eck, R.; and Park, C. 1972. A model of evolutionary change in proteins. In Dayhoff, M., editor 1972, *Atlas of Protein Sequence and Structure*, volume 5. National Biomedical Research Foundation: Silver Springs, MD.

Dayhoff, M.; Schwartz, R.; and Orcutt, B. 1978. A model of evolutionary change in proteins. In Dayhoff, M., editor 1978, *Atlas of Protein Sequence and Structure*, volume 5 (suppl. 3). National Biomedical Research Foundation: Silver Springs, MD. 345–358.

DeJong, G.F. and Mooney, R.J. 1986. Explanation-based learning: An alternative view. *Machine Learning* 1:145–176.

DeJong, G.F 1994. Learning to plan in continuous domains. *Artificial Intelligence* 65:71–141.

Devijver, P.A. and Kittler, J. 1982. *Pattern Recognition: A Statistical Approach.* Prentice Hall: Englewood Cliffs, NJ.

Dietterich, T.G. and Michalski, R. 1986. Learning to predict sequences. In Michalski, R.S.; Carbonell, J.G.; and Mitchell, T.M., editors 1986, *Machine Learning: An Artificial Intelligence Approach II.* Morgan Kaufmann: Los Altos, CA.

Dietterich, T.G.; London, B.; Clarkson, K.; and Dromey, G. 1982. Learning and inductive inference. In Cohen, P.R. and Feigenbaum, E.A., editors 1982, *The Handbook of Artificial Intelligence III.* Morgan Kaufmann.

Dietterich, T.G. 1986. Learning at the knowledge level. *Machine Learning* 1:287–316.

Dill, K. 1990. Dominant forces in protein folding. *Biochemistry* 29:7133–7155.

Donoho, S. and Rendell, L. 1996. Constructive induction using fragmentary knowledge. In *Proceedings of the International Conference on Machine Learning.* accepted.

Doolittle, R.F. 1981. Similar amino acid sequences: Chance or common ancestry? *Science* 214:149–159.

Doolittle, R.F. 1986. *Of Urfs and Orfs: A Primer on How to Analyze Devised Amino Acid Sequences.* Oxford University Press: Oxford.

Draper, N.R. and Smith, H. 1981. *Applied Regression Analysis.* Wiley.

Drastal, G.; Czako, G.; and Raatz, S. 1989. Induction in an abstraction space. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* 708–712.

Duda, R.O. and Hart, P.E. 1973. *Pattern Classification and Scene Analysis.* Wiley.

Falkenhainer, B.; Forbus, K.D.; and Gentner, D. 1989. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence* 41:1–63.

Finkelstein, A.V. and Ptitsyn, O.B. 1987. Why do globular proteins fit the limited set of folding patterns? *Progress in Biophysics and Molecular Biology* 50:171–190.

Fisher, R. 1936. The use of multiple measurements in taxonomic problems. *Annual Eugenics* 7:179–188.

Fisher, F.H. 1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2:139–172.

Fitch, W.M. and Smith, T.F. 1983. Optimal sequence alignments. *Proceedings of the National Academy of Sciences, USA* 80:1382–1386.

Fitch, W.M. 1970. Further improvements in the method of testing for evolutionary homology among proteins. *Journal of Molecular Biology* 49:1–14.

Flann, N.S. and Dietterich, T.G. 1986. Selecting appropriate representations for learning from examples. In *Proceedings of the Fifth National Conference on Artificial Intelligence.* 460–466.

Ford, K.M. and Hayes, P.J., editors 1991. *Reasoning Agents in a Dynamic World: The Frame Problem.* JAI Press: London.

Friedman, J. 1991. Multivariate adaptive regression splines. *Annals of Statistics* 19:1–141.

Fu, L. and Buchanan, B.G. 1985. Learning intermediate concepts in constraining a hierarchical knowledge base. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence.* 659–666.

Fu, K.S. 1982. *Syntactic Pattern Recognition and Applications.* Prentice-Hall: Englewood Cliffs, NJ.

Gaines, B. 1989. An ounce of knowledge is worth a ton of data. In *Proceedings of the Sixth International Workshop on Machine Learning.* 156–159.

Genesereth, M.R. and Nilsson, N.J. 1987. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann: Palo Alto, CA.

Ginsberg, A.; Weiss, S.M.; and Politakis, P. 1988. Automatic knowledge base refinement for classification systems. *Artificial Intelligence* 35:197–226.

Glasgow, J.I.; Fortier, S.; and Allen, F.H. 1993. Molecular scene analysis: Crystal structure determination through imagery. In Hunter, L., editor 1993, *Artificial Intelligence and Molecular Biology.* MIT Press: Cambridge, CA.

Gordon, D. and DesJardins, M. 1995. Evaluation and selection of biases in machine learning. *Machine Learning* 20:5–22.

Gotoh, O. 1982. An improved algorithm for matching biological sequences. *Journal of Molecular Biology* 162:705–708.

Gribskov, M.; McLachlan, A.D.; and Eisenberg, D. 1987. Profile analysis: Detection of distantly related proteins. *Proceedings of the National Academy of Sciences, USA* 84:4355–4358.

Gribskov, M.; Homyak, M.; Edenfield, J.; and Eisenberg, D. 1988. Profile scanning for three-dimensional structural patterns in protein sequences. *CABIOS* 4:61–66.

Harman, H.H. 1976. *Modern Factor Analysis.* University of Chicago Press: Chicago, IL.

Haussler, D. 1988. Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial Intelligence* 36:177–221.

Hayes-Roth, B.; Buchanan, B.; Lichtarge, O.; Hewett, M.; Altman, R.; Brinkley, J.; Cornelius, C.; Duncan, B.; and Jardetsky, O. 1986. Protean: Deriving protein structure from constraints. In *Fifth National Conference on Artificial Intelligence*. 904–909.

Henikoff, S. and Henikoff, J.G. 1993. Performance evaluation of amino acid substitution matrices. *Proteins* 17:49–61.

Hirsch, H. and Japkowicz, N. 1994. Bootstrapping training-data representations for inductive learning: A case study in molecular biology. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. 639–644.

Hirschberg, D.S. 1975. A linear space algorithm for computing maximal common subsequences. *Communication of the ACM* 18:341–343.

Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems.* University of Michigan Press: Ann Arbor, MI.

Holte, R.C. and Zimmer, R.M. 1989. A mathematical framework for studying representation. In *Proceedings of the Sixth International Workshop on Machine Learning.* Morgan Kaufmann: San Mateo, CA. 454–456.

Holte, R.C. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine Learning* 11:63–91.

Hong, S.J. 1994. Use of contextual information for feature ranking and discretization. Technical Report Research report RC19664, IBM Research Division; Yorktown Heights, NY.

Ioerger, T.R.; Rendell, L.; and Subramaniam, S. 1995. Searching for representations to improve protein sequence fold-class prediction. *Machine Learning* 21:151–175.

Jaffar, Joxan and Lassez, Jean-Louis 1987. Constraint logic programming. In *Fourth ACM Symposium on Principles of Programming Languages*. 111–119.

Jones, D. and Thornton, J. 1993. Protein fold recognition. *Journal of Computer-Aided Molecular Design* 7:439–456.

Jones, R. 1992. Protein sequence and structure comparison on massively parallel computers. *International Journal of Supercomputer Applications* 6:138–146.

Kabsch, W. and Sander, C. 1983. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometric features. *Biopolymers* 22:2577–2637.

Kadie, C.M. 1995. *SEER: Maximum-likelihood regression curves for learning-speed curves.* Ph.D. Dissertation, University of Illinois, Department of Computer Science.

Kidera, A.; Konishi, Y.; Oka, M.; Ooi, T.; and Scheraga, H.A. 1985. Statisitical analysis of the physical properties of the 20 naturally occuring amino acids. *Journal of Protein Chemistry* 4:23–54.

King, R.D.; Hirst, J.D.; and Sternberg, M.J.E. 1995. A comparison of artificial intelligence methods for modelling pharmaceutical qsars. *Applied Artificial Intelligence* 9:213–234.

Korf, R.E. 1980. Towards a model of representation changes. *Artificial Intelligence* 14:41–78.

Korf, R.E. 1982. A program that learns to solve rubik's cube. In *Proceedings of AAAI-82*. 164–167.

Koshland, D.E. 1958. Application of a theory of enzyme specificity to protein synthesis. *Proceedings of the National Academy of Sciences, USA* 44:98–104.

Langley, P.; Bradshaw, G.L.; and Simon, H.A. 1983. Redicovering chemistry with the bacon system. In Michalski, R.S.; Carbonell, J.G.; and Mitchell, T.M., editors 1983, *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann.

Lasters, I. 1990. The design of idealized alpha-beta barrels: Analysis of beta-sheet closure requirements. *Proteins* 7:249–256.

Lathrop, R.H. and Smith, T.F. 1996. Global optimum protein threading with gapped alignment and empirical pair score functions. *Journal of Molecular Biology* 255:641–665.

Lathrop, R.H.; Webster, T.A.; and Smith, T.F. 1987. Ariadne: Pattern-directed and hierarchical abstraction in protein structure prediction. *Communications of the ACM* 30:909–921.

Lathrop, R.; Webster, T.A.; Smith, T.F.; and Winston, P.H. 1990. Ariel: A massively-parallel symbolic learning assistant for protein structure/function. In Winston, P.H. and Shellard, S., editors 1990, *Artificial Intelligence at MIT: Expanding Frontiers.* MIT Press: Cambridge, MA.

Lathrop, R.; Webster, T.; Smith, R.; Winston, P.; and Smith, T. 1993. Integrating ai with sequence analysis. In Hunter, L., editor 1993, *Artificial Intelligence and Molecular Biology.* AAAI Press: Menlo Park, CA. 210–258.

Levitt, M. and Chothia, C. 1976. Structural patterns in globular proteins. *Science* 261:552–557.

Lim, V. 1974. Algorithms for proediction of alpha-helical and beta-structural regions in globular proteins. *Journal of Molecular Biology* 88:873–894.

Lipman, D.J. and Pearson, W.R. 1985. Rapid and sensitive protein similarity searches. *Science* 227:1435–1441.

Lloyd, J. 1988. *Foundations of Logic Programming.* Springer-Verlag.

Mansour, Y. 1994. Learning boolean functions via the fourier transform. In Roychodhury, V.P.; Siu, K-Y.; and Orlitsky, A., editors 1994, *Advances in Neural Computation.* Kluwer.

Matheus, C.J. and Rendell, L.A. 1989. Constructive induction on decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. 645–650.

Matheus, C.J. 1989. *Feature Construction: An Analytic Framework and an Application to Decision Trees.* Ph.D. Dissertation, University of Illinois, Department of Computer Science.

McCammon, J. and Harvey, S. 1987. *Dynamics of Proteins and Nucleic Acids.* Cambridge University Press: New York.

McLachlan, A.D. 1971. Tests for comparing related amino-acid sequences: Cytochrome c and cytochrome c551. *Journal of Molecular Biology* 61:409–424.

McLachlan, A.D. 1972. Gene duplication in carp muscle calcium-binding protein. *Nature New Biology* 240:83–85.

Michalski, R.S. and Stepp, R.E. 1983. Learning from observation: Conceptual clustering. In Michalski, R.S.; Carbonell, J.G.; and Mitchell, T.M., editors 1983, *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann.

Michalski, R. 1983. A theory and methodology of inductive learning. *Artificial Intelligence* 20:111–161.

Miller, S.; Janin, J.; Lesk, A.M.; and Chothia, C. 1987. Interior and surface of monomeric proteins. *Journal of Molecular Biology* 196:641–656.

Minsky, M.L. and Papert, S. 1969. *Perceptrons: An Introduction to Computational Geometry.* MIT Press: Cambridge, MA.

Minsky, M. 1963. Steps toward artificial intelligence. In Feigenbaum, E.A. and Feldman, J., editors 1963, *Computers and Thought.* McGraw Hill: New York.

Mitchell, T. 1980. The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers University, Department of Computer Science.

Mitchell, T. 1982. Generalization as search. *Artificial Intelligence* 21:203–226.

Mooney, R.J. and Ourston, D. 1994. A multistrategy approach to theory refinement. In Michalski, R.S. and Tecuci, G., editors 1994, *Machine Learning: A Multistrategy Approach*, volume 4. Morgan Kaufmann: San Fransisco.

Muggleton, S. and Buntine, W. 1988. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning.* 218–229.

Myers, E.W. and Miller, W. 1988. Optimal alignments in linear space. *CABIOS* 4:11–17.

Needleman, S. and Wunsch, C. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48:443–453.

Nei, M. 1987. *Molecular Evolutionary Genetics.* Columbia University Press: New York.

Neidhart, D.J.; Kenyon, G.L.; Gerlt, J.A.; and Petsko, G.A. 1990. Mandelate racemase and muconate lactonizing enzyme are mechanistically distinct and structurally homologous. *Nature* 347:692–694.

Nell, L.J.; McCammon, J.A.; and Subramaniam, S. 1992. Anti-insulin antibody. structure and conformation i. molecular modeling and mechanics. *Biopolymers* 32:11–21.

Newell, A. and Simon, H.A. 1972. *Human Problem Solving.* Prentice Hall: Englewood Cliffs, NJ.

Newell, A. 1969. Heuristic programming iii: Structured problems. In Aronofsky, J.S., editor 1969, *Progress in Operations Research*, volume 3. Wiley: New York.

Norton, S.W. 1989. Generating better decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* 800–805.

Overington, J.; Donnelly, D.; Johnson, J.S.; Sali, A.; and Blundell, T. 1992. Environment-specific amino acid substitution tables: Tertiary templates and prediction of protein folds. *Protein Science* 1:216–226.

Packard, N.H. 1989. Genetic learning algorithm for the analysis of complex data. Technical Report CCSR-89-10, University of Illinois, Center of Complex Systems.

Pagallo, G. and Haussler, D. 1990. Feature discovery in empirical learning. *Machine Learning* 5:71–99.

Pascarella, S. and Argos, P. 1992. A data bank merging related protein structures and sequences. *Protein Engineering* 5:121–137.

Pazzani, M.J. 1988. Integrated learning with incorrect and incomplete theories. In *Proceedings of the Fifth International Conference on Machine Learning.* 291–297.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley: Reading, MA.

Perutz, M.F.; Kendrew, J.C.; and Watson, J.D. 1965. Structure and function of haemoglobin ii: Some relations between polypeptide congifuration and amino acid sequence. *Journal of Molecular Biology* 13:669.

Ponder, J.W. and Richards, F.M. 1987. Tertiary templates for proteins: Use of packing criteria in the enumeration of allowed sequences for different structural classes. *Journal of Molecular Biology* 193:775–791.

Porter, B.W.; Bareiss, R.; and Holte, R.C. 1990. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence* 45:229–263.

Qian, N. and Sejnowski, T.J. 1988. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology* 202:865–884.

Quinlan, J.R. 1983. Learning efficient classification procedure and their application to chess end-games. In Michalski, R.S.; Carbonell, J.G.; and Mitchell, T.M., editors 1983, *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann.

Quinlan, J.R. 1986. The effect of noise on concept learning. In Michalski, R.S.; Carbonell, J.G.; and Mitchell, T.M., editors 1986, *Machine Learning: An Artificial Intelligence Approach II.* Morgan Kaufmann: Los Altos, CA.

Quinlan, J.R. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann: Palo Alto, CA.

Rabiner, L.R. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–286.

Ragavan, H. and Rendell, L. 1991. Relations, knowledge, and empirical learning. In *Eighth International Workshop on Machine Learning*. 188–192.

Ragavan, H. and Rendell, L. 1993. Lookahead feature construction for learning hard concepts. In *Proceedings of the Tenth International Conference on Machine Learning*. 252–259.

Ragavan, H.; Rendell, L.; Shaw, M.; and Tessmer, A. 1993. Complex concept acquisition through directed search and feature caching. In *Proceeding of the Thirteenth International Joint Conference on Artificial Intelligence*. 946–951.

Rajamoney, S. 1990. A computational approach to theory revision. In Shrager, J. and Langley, P., editors 1990, *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann: San Mateo, CA.

Ramachandran, G.N. and Sasisekharan, V. 1968. Conformation of polypeptides and proteins. *Advances in Protein Chemistry* 23:283–437.

Rendell, L.A. and Cho, H.H. 1990. Empirical concept learning as a function of concept character. *Maching Learning* 5:267–298.

Rendell, L. and Ragavan, H. 1993. Improving the design of induction methods by analyzing algorithm functionality and data-based complexity. In *Proceeding of the Thirteenth International Joint Conference on Artificial Intelligence*. 952–958.

Rendell, L. and Seshu, R. 1990. Learning hard concepts through constructive induction: Framework and rationale. *Computational Intelligence* 6:247–270.

Rendell, L.A. 1985. Substantial constructive induction using layered information compression: Tractable feature formation in search. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. 650–658.

Rendell, L. 1987. Representation and models for concept learning. Technical Report UIUCDCS-R-87-1324, Department of Computer Science, University of Illinios.

Richards, F. 1992. Folded and unfolded proteins: An introduction. In Creighton, T., editor 1992, *Protein Folding*. Freeman: New York. 1–58.

Richardson, J.S. and Richardson, D.C. 1989. Principles and patterns of protein conformation. In Fasman, G.D., editor 1989, *Prediction of Protein Structure and the Principles of Protein Conformation*. Plenum Press: New York. 1–98.

Richardson, J.S. 1981. The anatomy and taxonomy of protein structure. *Advances in Protein Chemistry* 34:167–336.

Rumelhart, D.E.; Hinton, G.E.; and Williams, R.J. 1986. Learning internal representations by error propagation. In Rumelhart, D.E. and McClelland, J.L., editors 1986, *Parallel Distributed Processing*, volume 1. MIT Press: Cambridge, MA.

Russell, S.J. and Grosof, B.N. 1987. A declarative approach to bias in concept learning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. 505–510.

Samuel, A.L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* 3:210–229.

Sander, C. and Schneider, R. 1991. Database of homology-derived protein structures and the structural meaning of sequence alignment. *Proteins* 9:56–68.

Sankoff, D. and Kruskal, J. 1983. *Time Warps, String Edits, and Macromolecules.* Addison-Wessley: Reading, MA.

Saxena, S. 1991. On the effect of instance representation on generalization. In *Proceedings of the Eighth International Machine Learning Workshop.* 198–202.

Schaeffer, C. 1993. Selecting a classification method by cross-validation. *Machine Learning* 13:135–142.

Schlimmer, J.C. 1987. Learning and representation change. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence.* 511–515.

Schulz, G.E. and Schirmer, R.H. 1979. *Principles of Protein Structure.* Springer-Verlag: New York.

Seshu, R.; Rendell, L.; and Tcheng, D. 1988. Managing constructive induction using subcomponent assessment and mulitple-objective optimization. In *Proceedings of the First International Workshop on Change of Representation and Inductive Bias.* 293–305.

Seshu, R. 1989. Solving the parity problem. In *Proceedings of the European Working Session on Learning.* 263–271.

Simon, H.A. and Lea, G. 1973. Problem solving and rule induction: A unified view. In Gregg, L.W., editor 1973, *Knowledge and Cognition.* Lawrence Erlbaum Associates: Potomac, MD. 105–127.

Sippl, M.J and Weitckus, S. 1992. Detection of native-like models for amino acid sequences of unknown three-dimensional structures in a database of known protein conformations. *Proteins* 13:258–271.

Smith, R.F. and Smith, T.F. 1990. Automatic generation of primary sequence patterns from sets of related protein sequences. *Biochemistry* 87:118–122.

Smith, T.F. and Waterman, M.S. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147:195–197.

Smith, T.F.; Waterman, M.S.; and Burks, C. 1985. The statistical distribution of nucleic acid similarities. *Nucleic Acids Research* 13:645–656.

Srinivasan, R. 1976. Helicial length distribution from protein crystallographic data. *Industrial Journal of Biochemistry and Biophysics* 13:192–193.

Steele, J.M. 1982. Long common subsequences and the proximity of two random strings. *SIAM Journal of Applied Mathematics* 42:731–737.

Sternberg, M.J.E. and Thornton, J. 1977. On the conformation of proteins: An analysis of beta-pleated sheets. *Journal of Molecular Biology* 110:285–296.

Stolorz, P.; Lapedes, A.; and Xia, Y. 1992. Predicting protein secondary structure using neural net and statistical methods. *Journal of Molecular Biology* 225:363–377.

Stone, M. 1974. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society* 36:111–147.

Stryer, L. 1988. *Biochemistry.* Freeman: New York.

Tanford, C. 1973. *The Hydrophobic Effect.* Wiley: New York.

Taylor, W.R. and Orengo, C.A. 1989. Protein structure alignment. *Journal of Molecular Biology* 208:1–22.

Taylor, W.R. 1986. Identification of protein sequence homology by consensus template alignment. *Journal of Molecular Biology* 188:233–258.

Tcheng, D.K.; Lambert, B.L.; Lu, S.C.Y.; and Rendell, L.A. 1989. Building robust learning systems by combining induction and optimization. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* 806–812.

Thornton, S. and Thornton, W. 1990. Beta-turns and their distortions: A proposed new nomenclature. *Protein Engineering* 3:479–493.

Towell, G.; Shavlik, J.; and Noordewier, M. 1990. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence.* 861–866.

Utgoff, P. 1986. Shift of bias for inductive concept learning. In Michalski, R.; Carbonell, J.; and Mitchell, T., editors 1986, *Machine Learning: An Artificial Intelligence Approach II.* Morgan Kaufmann: Los Altos, CA. 107–148.

Valiant, L.G. 1984. A theory of the learnable. *Communcations of the ACM* 27:1134–1142.

Watson, J.D. 1990. The human genome project: Past, present, and future. *Science* 248:44–49.

Weiss, S.M. and Kapouleas, I. 1989. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Morgan Kaufmann. 781–787.

Weiss, S.M. and Kulikowski, C.A. 1991. *Computer Systems that Learn.* Morgan Kaufmann: San Mateo, CA.

White, F.H. 1961. Regneration of native secondary and tertiary structures by air oxidation of reduced ribonuclease. *Journal of Biological Chemistry* 236:1353–1360.

Wilkins, D.C. 1990. Knowledge base refinement as improving an incomplete and incorrect domain theory. In Kodratoff, Y. and Michalski, R.S., editors 1990, *Machine Learning, an Artificial Intelligence Approach, Volume III.* Morgan Kaufmann: San Mateo, CA. 493–513.

Winston, P. 1975. Learning structural descriptions from examples. In Minsky, M.L., editor 1975, *The Psychology of Computer Vision.* McGraw Hill: NewYork.

Zhang, X. and Waltz, D. 1993. Developing hierarchical representations for protein structures: An incremental approach. In Hunter, L., editor 1993, *Artificial Intelligence and Molecular Biology.* MIT Press: Cambridge, CA.

Zubay, G. 1983. *Biochemistry.* Addison-Wesley: Reading, MA.

Vita

Thomas Richard Ioerger was born on November 22, 1967 in Houston, TX. He graduated with honors from The Pennsylvania State University in 1989, getting his B.S. degree in Molecular and Cell Biology. As an undergraduate, he did research in the laboratory of Dr. Teh-hui Kao, focusing on computer analysis of DNA sequences for a protein involved in plant reproduction. His honor's thesis was titled "Molecular Evolution of Self-incompatibility in the Solanaceae Family." In 1990, he was awarded a National Science Foundation Graduate Fellowship in Computer Science and entered graduate school in the Department of Computer Science at the University of Illinois in Urbana-Champaign. He received his M.S. degree in 1992 for research on the indeterminacy problem in spatial reasoning. The title of his master's thesis was "A Method of Default Reasoning for Reasoning about Spatial Prepositions." His doctoral research involved an interdisciplinary study of machine learning and computational biology. He received his Ph.D. in 1996, with a dissertation titled "Change of Representation in Machine Learning, and an Application to Protein Structure Prediction." This work was conducted with Dr. Larry Rendell in the Department of Computer Science at the University of Illinois and Dr. Shankar Subramaniam in the Department of Biophysics at the University of Illinois. In 1996, he accepted a position as Assistant Professor in the Department of Computer Science at Texas A&M University in College Station, TX. His research interests include artificial intelligence, machine learning, cognitive science, and computational biology.