LEARNING TO PREDICT SEQUENCES

by

Tom Dietterich
Ryszard S. Michalski

Report #939, Department of Computer Science, University of Illinois, Urbana, Illinois, February, 1985.

File No. UIUCDCS-F-85-939

Learning to Predict Sequences

Thomas G. Dietterich
Oregon State University
Corvallis, OR

Ryszard S. Michalski*
Massachusetts Institute of Technology
Cambridge, MA

February 1985

ISG 85-9

* On leave of absence from the University of Illinois at Urbana-Champaign.

This work was supported in part by the National Science Foundation under grant NSF DCR 84-06801 and the Office of Naval Research under grant N00014-82-K-0186.

Table of Contents

List of Figures

Abstract

This chapter considers the problem of discovering a rule characterizing a given sequence of events (objects) and able to predict a plausible continuation of the sequence. This prediction is non-deterministic because the rule doesn't necessarily tell exactly which events must appear next in the sequence but rather determines a set of plausible next events. It is assumed that the individual events in the sequence are characterized by a set of attributes and that the next event depends solely on the values of the attributes for the previous events in the sequence. The attributes are either initially given or are derived from the initial ones through a chain of inferences. Three basic rule models are employed to guide the search for a sequence-generating rule: decomposition, periodic, and disjunctive normal form (DNF). The search process involves simultaneously transforming the initial sequences to *derived* sequences and instantiating general rule models to find the best match between the instantiated model and the derived sequence. A program, called SPARC/E, is described that implements most of the methodology as applied to discovering sequence generating rules in the card game Eleusis. This game, which attempts to model the process of scientific discovery, is used as a source of examples illustrating the performance of SPARC/E.

Key terms: Machine learning, sequence extrapolation, inductive inference, part-to-whole induction, data transformation, model-directed learning, Eleusis.

1 Introduction

Inductive learning—that is, learning by generalizing specific facts or observations—is a fundamental strategy by which we acquire knowledge about the world. Computer models of inductive learning have been studied from the AI perspective for several years now, and one result of this research has been the identification of several different kinds of inductive learning. At least three different types have been studied. These are (a) *instance-to-class induction*, (b) *part-to-whole induction*, and (c) *conceptual clustering*.

Instance-to-class induction has received the most attention. Here, the learning system is presented with independent instances representing some class, and the task is to induce a general description of the class. The instances can be specific physical objects, actions, processes, images, and so on. The learned class description (also called the concept description) can be used to classify new instances whose correct class is not known.

An example of this type of learning problem is one of determining diagnostic rules from a set of diagnosed cases of diseases. For example, Michalski and Chilausky [1980] describe a learning program, AQ11, that is

presented with a set of independent training instances, each of which is an example of a soybean plant with a given disease. The AQ11 program then induces a general description of that disease. This description can be applied to diagnose the occurrence of this disease in other soybean plants. From several hundred such training instances covering 19 different soybean diseases, AQ11 has inferred a set of 19 diagnostic rules. Several other researchers have investigated instance-to-class learning problems (e.g., [Winston, 1970], [Buchanan and Mitchell, 1978], [Mitchell, Utgoff, and Banerji, 1983]). Reviews of various methods for such instance-to-class induction appear in [Michalski, Carbonell, and Mitchell (eds.), 1983] and [Dietterich, et al., 1982].

The second type of inductive learning—part-to-whole induction—has received less attention. Part-to-whole induction involves constructing a description of a whole object by observing only selected parts of it. For example, given a set of fragments of a scene, the problem is to hypothesize the description of the whole scene. An important part-to-whole induction problem is one of discovering a description of a sequence of objects, where the "part" consists of the first n elements of the sequence and the "whole" is a sequence-generating rule that can predict possible continuations of the given part of the sequence.

This type of part-to-whole induction problem has been studied in the past under the name of "sequence extrapolation" or "letter-sequence prediction." Simon and Kotovsky [Simon & Kotovsky, 1963; Simon, 1972; Kotovsky & Simon, 1973], for example, study problems in which a program (or a person) is given partial sequences such as:

A B X B C W C D V ...

and asked to predict the next few letters in the sequence. Their program does this by first finding a sequence-generating rule and then applying that rule to predict the continuation of the sequence. In this case, the rule might state that the sequence is a periodically repeating subsequence of three letters in which the first two letters are successors of the letter appearing in the previous period, while the third letter is the predecessor of the corresponding letter in the previous period. Related work on this type of learning problem has been performed by Solomonoff [1964], Hedrick [1976], and Hofstadter [1983, In Press].

The third type of learning problem, conceptual clustering, has received very little attention. We mention it here only for completeness. Clustering problems arise when several objects (or situations) are presented to a learner, and the learner must invent classes into which the objects can be usefully grouped. An example of such a problem is learning sound systems in spoken language. The human ear is capable of distinguishing among a wide variety of spoken sounds. However, any given human language groups all of these sounds into a relatively small number of equivalence classes (roughly 50). All sounds within a given class are regarded as being identical for purposes of communication. Recently, Michalski [1980] and Michalski and Stepp [1983] developed a method and a computer program, CLUSTER, for *conjunctive conceptual clustering* that can solve such learning problems.

This chapter presents further research on the second type of inductive learning problem, that is, part-to-whole induction. We are particularly interested in sequence prediction problems that are much more complex than letter-series prediction. Letter-series prediction is a very simple problem for two reasons. First, in letter series, each object in the sequence has only one attribute—its name. Second, the desired sequence-prediction rule is deterministic, because it is assumed that there is only one legal continuation of the sequence. This chapter presents a method for discovering sequence-prediction rules in cases where the objects are described by several relevant attributes and where the sequence-prediction rule is non-deterministic. This type of learning problem is called a *Nondeterministic Prediction Problem*, or an NDP problem.

In an NDP problem, the learner is presented with a finite sequence of events. Each event is characterized by the values of a number of discrete-valued attributes. The goal is to find a *sequence-generating rule* that, given the first k events, states the values of the attributes[1] that must be true of event $k+1$. Since the sequence-generating rule need only state values for *some* of the attributes, the rule may not necessarily predict a unique event $k+1$. This is what makes the rule non-deterministic. Because only a partial description of the original sequence is sought and because the description may involve new attributes not present in the initial set, a very large number of hypotheses may need to be examined. This makes this problem much more difficult to solve than previously-studied letter-series extrapolation problems.

The card game Eleusis [Abbott, 1977; Gardner, 1977] presents a very general kind of non-deterministic sequence prediction problem. We will use examples from this game to illustrate the proposed general methodology for discovering rules for event sequences.

1.1 Eleusis: An Exemplary Non-Deterministic Prediction Problem

An interesting NDP problem arises in the card game Eleusis, invented by Robert Abbott [Abbott, 1977; Gardner, 1977]. Eleusis is an inductive game in which players attempt to discover a generating rule (known only to the dealer) for a sequence of cards. This "secret rule" is invented and recorded by the dealer before the game. Each player, in his or her turn, adds one card to the sequence, and the dealer indicates whether the card is a correct (or incorrect) extension of the sequence (i.e., satisfies or does not satisfy the secret sequence-generating rule). Players who play incorrectly are penalized by having additional cards added to their hands. The goal of each player is to get rid of all of the cards in his hand, which is only possible if correct cards are played. The cards played during the game are displayed in the form of a layout in which the correct cards form the "main line" and incorrect cards form "side lines" branching down from the main line at the card that

---

[1] It is assumed that the rule is expressed in terms of attributes that are either observable attributes of objects present in the sequence up to the moment when a new object is generated or attributes that can be derived from such observable attributes by some known inference rules.

they followed. Figure 1 shows a typical Eleusis layout for the sequence-generating rule "Play alternating red and black cards." In this game, the 3 of hearts was played first, followed by a 9 of spades, and a Jack of diamonds. All of these were correct. Following the Jack, a 5 of diamonds was played. It appears on a sideline below the Jack, because it was not a correct extension of the sequence. (At this point, a black card is required.) The 4 of clubs was then correctly played, and so on.

```
Main line:   3H  9S  JD  4C  JD  2C 10D  2C  5H
Side lines:          5D      AH  AS  8H
                             8H 10S  7H
                             QD      10H
```

Figure 1: A sample Eleusis layout

Below are several examples of sequence-generating rules for Eleusis:

- If the last card was a spade, play a heart; if last card was a heart, play diamonds; if last was diamond, play clubs; and if last was club, play spades.
- The card played must be one point higher than or one point lower than the last card.
- If the last card was black, play a card higher than or equal to that card; if the last card was red, play lower or equal.
- Play alternating even and odd cards.
- Play strings of cards such that each string contains cards all in the same suit and has an odd number of cards in it.

There are four important points to note about this game. First, observe that an Eleusis rule typically allows any of several cards to be played legally after each card. Hence, Eleusis provides an instance of the non-deterministic prediction problem.

Second, notice that the rules employ descriptors or terms that do not appear in the input sequence. The input data provides only the suit and rank (value) of each card and its position in the sequence. The sequence-generating rules, however, may include such terms as "even", "odd", "black", "red", and "strings of cards such that each string contains cards all in the same suit and has an odd number of cards in it." The learning program must bridge this gap between the terms appearing in the input sequence and the terms needed for expressing the rules. To bridge this gap, one has to solve what is called the *description space transformation problem.*

The third point is that several different logical forms are employed to express the rules. Some rules take the form of a set of if-then rules: "If the last card was a spade, play a heart; ..." Other rules are stated as simple disjunctions: "The card played must be one point higher than or one point lower than the last card." And still others describe the layout as a periodically repeating sequence: "Play alternating even and odd cards". The learning program must have the capacity to create descriptions that capture these different logical forms. Our approach to the solution is to divide different sequence-generating rules into a few general classes according to the logical form of the rules. Each class is represented by an abstract *model*, or logical schema, which can be parameterized and then instantiated to yield a particular sequence-generating rule. For this reason, we call this method a *multiple model learning method.*

Finally, it should be noted that the space of possible Eleusis rules is very large. Indeed, there is in principle no limit to the number of secret rules. However, to make the game interesting to human players, Eleusis has a point-scoring scheme that encourages the dealer to choose only fairly simple sequence-generating rules. SPARC/E is capable of representing more than 10^{137} rules[2].

After these remarks, we can now state the three main problems addressed in this chapter. They are

- Transforming the original description space to aid induction,
- Applying multiple rule models to discover specific sequence-generating rules, and
- Developing a strategy and overall system architecture for learning with multiple models.

In the next two subsections, we present general descriptions of the first two of these problems and describe how they have been solved in other systems. Discussion of the overall system architecture is postponed until section 4.

---

[2] This estimate is based on computing the space of all syntactically legal VL_{22} conjunctive statements (see section 3) containing the following set of descriptors (each descriptor is followed by the number of elements in its value set and the number of possible selectors that can be formed using those elements): SUIT (4,9), RANK (13, 91), COLOR (2,3), FACEDNESS (2,3), PARITY (2,3), PRIMENESS (2,3), RANKMOD3 (3,7), D-SUIT01 (4,9), D-SUIT02 (4,9), D-RANK01 (25,300), D-RANK02 (25,300), S-RANK01 (25,300), S-RANK02 (25,300), D-COLOR01 (2,3), D-COLOR02 (2,3), D-FACEDNESS01 (2,3), D-FACEDNESS02 (2,3), D-PARITY01 (2,3), D-PARITY02 (2,3), D-PRIMENESS01 (2,3), D-PRIMENESS02 (2,3), D-RANKMOD3-01 (3,7), D-RANKMOD3-02 (3,7). The SUIT and RANKMOD3 descriptors are cyclically ordered, while the RANK descriptors are interval descriptors. All others are nominal. In a decomposition rule with a lookback of $L = 2$, the first seven descriptors appear three times—once for each card. Hence, the total number of possible conjuncts is $(9*91*3*1*3*3*7)^3 * (9*300*300*3*3*3*7)^2 = 2.11221*10^{34}$. If there are four conjuncts in a rule, then we obtain $[2.11221*10^{34}]^4 = 1.99*10^{137}$.

1.2 Transforming the Description-Space

The problem of transforming the initial problem description arises in any domain where the given data (e.g., the training instances in concept learning) are observations or measurements that do not include the information directly relevant to the task at hand. For example, in character recognition, the input typically consists of a matrix of light intensities representing a character, but the relevant information includes position-invariant properties of letters such as the presence of a line on the left or right of a character, occurrence of line endings, closed contours, and so on (e.g., [Karpinski and Michalski, 1966]). These position-invariant properties can be made explicit by applying description-space transformations to the raw data.

An example of a learning program that performs description space transformations is INTSUM, which is part of the Meta-DENDRAL system [Buchanan and Mitchell, 1978]. INTSUM is presented with raw training instances in the form of chemical structures (graphs) and associated mass spectra, represented as fragment masses and their intensities. For each fragment in the mass spectrum, INTSUM must determine the bonds that could have broken to produce that fragment. A simple mass spectrometer simulator is used to develop these hypothesized bond breaks. Each of the resulting transformed training instances has the form of a chemical structure and a set of bonds that broke when that structure was placed in the mass spectrometer. It is this information that is provided to the remaining parts of the Meta-DENDRAL system (programs RULEGEN and RULEMOD).

In character recognition programs and in Meta-DENDRAL, the data transformations are fixed in advance. In many learning programs, however, the proper transformations are not known *a priori*. In such cases, a learning system needs to select or invent appropriate descripton-space transformations.

The type of description-space transformation performed by a program is a useful criterion for characterizing learning methods. The simplest learning methods (e.g., linear regression) determine only the coefficients for an *a priori* determined, fixed set of variables arranged in a predetermined expression. More sophisticated are learning algorithms, such as the Λ^q algorithm [Michalski, 1969, 1972] and the candidate elimination algorithm [Mitchell, 1978, 1983], that are able to determine which attributes are relevant and how they should be combined. Another level of sophistication is obtained when a learning program applies a set of predetermined transformations to the data prior to inductive generalization (e.g., Soloway [1981], Meta-DENDRAL [Buchanan and Mitchell, 1978]). These programs augment the basic inductive algorithms by applying a set of predetermined transformations to the data prior to inductive generalization. The next step of difficulty is represented by learning algorithms that select description space transformations under the guidance of special heuristics. Very few researchers have addressed this problem (e.g., [Lenat, 1983; Michalski, 1983]). The most sophisticated algorithms currently envisioned—but not yet developed—would

be capable of discovering new data transformations. Figure 2 shows this spectrum of inductive learning problems.

1. Determine coefficients
2. Select relevant variables and combine
3. Apply predetermined transformations
4. Select transformations under heuristic guidance
5. Discover new transformations

Figure 2: Spectrum of learning problems in increasing order of difficulty

The SPARC/E method presented in this chapter falls under category 4 of Figure 2, as it selects transformations under heuristic guidance. The program has available four general classes of description space transformations (see Section 2) from which it selects the appropriate ones to apply under the guidance of domain-specific heuristics.

1.3 Learning With Multiple Models

The second major problem that arises in the Eleusis prediction problem is concerned with using multiple description models in the process of inductive learning. This problem has not received much attention in previous AI research. Almost all existing learning systems employ only a *single model* for determining the space of possible output descriptions (hypotheses). Many systems, for example, use conjunctive models to represent concepts, that is, they assume that the learned concept will be expressed as a conjunction of predicates. By constraining the search to consider only conjunctive descriptions, the learning problem is greatly simplified.

A more general approach, employed by Michalski [1969, 1972], constrains descriptions to be in disjunctive normal form with *fewest* conjunctive descriptions. The induction algorithm finds first one conjunction, then another, and so on until all of the training instances are covered. Meta-DENDRAL employs a fairly elaborate simulator of the operation of the mass spectrometer to guide its search for conjunctive cleavage rules [Buchanan and Mitchell, 1978]. In general, current learning systems use a single model, and very few authors have made their models explicit.

One researcher who has employed multiple models is Persson [1966]. He applied four different models to the problem of extrapolating number- and letter-sequences. Briefly, these models were the following:

1. a model that computes the coefficients and the degree of a polynomial by applying Newton's forward-difference formula (the degree can be arbitrarily large);

2. an extended model that discovers exponential rules of the form AB^C, where A is a polynomial of degree 4 or less, and B and C are polynomials of degree 1 or less (i.e., B and C are of the form $ax + b$);

3. a simple periodic model for periods of length 2 (i.e., intertwined sequences); and

4. a generalization of the Kotovsky and Simon model for Thurstone letter-series that can discover simple periodic and segmented sequence-generating laws.

These models are applied by the program in a rather unusual learning situation in which the program is given a *sequence* of sequence-extrapolation problems. Thus, in addition to attempting to solve each individual sequence-extrapolation problem, Persson's program tries to predict the *kind* of sequence-prediction problem that it will next receive.

Persson's work shows the value of employing multiple description models to search for sequence-generating rules. The major limitation of Persson's approach, however, is that it is specific to number- and letter-sequence prediction. His methods cannot solve the more general prediction problems described in this chapter in which events have multiple attributes (both numerical and non-numerical) and the sequences are characterized by nondeterministic logic-based sequence-prediction rules.

One can conceive a spectrum of five model-based learning methods (see Figure 3). The simplest approach is to use a single model. This has been the common approach in AI thus far. The next step is to provide a learning program with a set of models from which it would choose the most appropriate ones. This is the approach used by Persson. The third level of sophistication would be to have the program generate a predetermined set of models, just as the learning program applies a predetermined set of data transformations. This method could be improved further by having the program decide which models to generate on the basis of special heuristics. Finally, an even more sophisticated program would be able to invent new models and apply them to guide the learning process.

The approach described in this chapter searches a predetermined space of possible models in a depth-first fashion, and hence, falls under point 3 of Figure 3.

The main theoretical contributions of this research include the development of techniques for (a) selecting description-space transformations, (b) applying of multiple description models, and (c) matching instantiated models to the transformed sequences using a bidirectional search.

1. Single model
2. Selection from a few models
3. Predetermined generation of models
4. Heuristically-guided generation of models
5. Discovery of new models

Figure 3: Spectrum of model-based methods in increasing difficulty

1.4 Overview of Solution

This section gives an overview of our approach to solving NDP problems. As described above, the learning program is given an input sequence of events. We assume that a sequence of events is given and that the task is to find a non-deterministic sequence-generating rule characterizing the input sequence and able to predict its plausible continuation. In the proposed solution, the learning program is supplied with various operators for sequence transformation and with specifications of different rule models. (The implemented method employs four sequence-transformation operators and three rule models.)

The sequence-transformation operators are repeatedly applied to the input sequence to yield *derived sequences* in which additional facts about the sequence are made explicit. This is a bottom-up process of elaborating and reformulating the data. Simultaneously, through a top-down process, the general rule models are specialized by filling in various parameters and formulas to obtain specific sequence-generating rules. Lastly, the learning program applies three model-fitting algorithms (one for each model) to fit the partially-refined model to the transformed data. Thus, the learning program conducts both a bottom-up elaboration of the data and a top-down specialization of the rule models until one of the model-fitting algorithms can be applied to find a match between the elaborated sequence and some specialized rule model. One or more resulting specialized rules are output as candidate sequence-generating rules.

Figure 4 illustrates this process schematically. The top-down model specialization process occurs in parallel with the bottom-up data transformation process, thus constituting a kind of bi-directional search.

Model instantiation, as used in this chapter, is an extension of the well-known AI technique of schema instantiation. Schema instantiation has been applied, for example, by Schank and Abelson [1975] to interpret natural language, by Engelmore and Terry [1979] to interpret X-ray diffraction data in protein chemistry, and by Friedland [1979] to plan genetics experiments. Model instantiation differs from schema instantiation in the complexity of the instantiation process. Model instantiation involves not only filling in predetermined

Figure 4: Schematic description of the rule discovery process

slots or substituting constants for variables, but also synthesizing a logical formula of an assumed type. For example, in order to instantiate each of the three models described below, the program must synthesize a conjunction of predicates or a disjunction of such conjunctions that satisfies certain constraints. Model-instantiation methods share with schema-instantiation methods the advantage that they are efficient and also effective with noisy and uncertain data. The constraints provided by the models (or schemas) drastically reduce the size of the space that the program must search.

The principal disadvantage of model- and schema-instantiation methods is that they require substantial amounts of domain knowledge to be built into the program. In order to keep this domain knowledge explicit and easily modified, we employ a ring architecture in the design of the learning program, as described in section 4. This architecture facilitates the application of the system to a variety of problems by simplifying the process of changing the domain-specific parts of the program.

The remaining sections of this chapter discuss the following:

1. the methods used for representing and transforming the initial training instances,

2. the techniques for representing the models and sequence-generating rules, and finally,

3. the details of the program SPARC/E, which implements most of the described methodology. The model-fitting algorithms are presented and the program is illustrated by a few selected examples of its operation when applied to the inductive card game Eleusis.

2 Transforming the Original Description Space

Now that we have defined the problem to be solved (the NDP problem) and sketched the solution, we launch into the details of that solution. We start by presenting the language for describing the original sequences and the transformation operators for changing this initial representation into a form more amenable for discovering sequence-generating rules.

2.1 Representing the Initial Sequence

A sequence of objects is represented as an indexed list[3]

$$\langle q_1, q_2, ..., q_k \rangle$$

Each object q_i is described by a set of attributes (also called *descriptors*) $f_1, f_2, ..., f_n$, which can be viewed as functions mapping objects into attribute values. To state that attribute f_i of object q_j has value r, we write

$$[f_i(q_j)=r].$$

This expression is called a *selector*. For example, if f_i is *color* and r is *red*, then the selector

$$[\text{COLOR}(q_j)=\text{red}]$$

states that the color of the j-th object in the sequence is red.

Each attribute is only permitted to take on values from a finite value set called the *domain*, $D(f_i)$, of that attribute. This constraint is part of the background knowledge that has to be given to the program. For example, in a deck of cards, the domain of the SUIT attribute is {clubs, diamonds, hearts, spades}. Additional knowledge about the domain set can be represented. In particular, the domain set may be linearly ordered, cyclically ordered (i.e., have a circular, wrap-around ordering), or tree ordered. We will see below how these domain orderings are applied to the problem of representing cards in an Eleusis game.

A complete initial description of a single object, q_j, called an *event*, is an expression giving the values for all of the attributes applicable to q_j. This is usually written as a conjunction of selectors:

$$[f_1(q_j)=r_1][f_2(q_j)=r_2]...[f_n(q_j)=r_n].$$

It can also be represented as a vector of attribute values:

$$(r_1, r_2, ..., r_n).$$

This vector notation suggests that each object description can be viewed as a point in the *event space* E:

$$E = D(f_1) \times D(f_2) \times ... \times D(f_n)$$

where $D(f_i)$ is the domain of attribute f_i. This event space contains all possible events.

---

[3] A summary of the notational conventions used in this chapter appears in Appendix I.

A complete description of the initial sequence is a list of conjunctions of selectors (or alternatively, a list of attribute vectors)—one conjunction for each object in the sequence. Each sequence description can thus be viewed as a trajectory in the event space, E. The space of all possible sequences is the set of all possible trajectories of events in E. It is important to note, however, that because of the discreteness of the space, these trajectories are not continuous. That is, two adjacent events in the sequence may not be "close" in the event space.

2.2 Transformation Operators

As we mentioned in section 1, it is often necessary to transform the initial sequence into a *derived sequence* in order to facilitate the discovery of sequence-generating rules. Such a data transformation can be viewed as a mapping T from one set of sequences S, containing objects Q, described by attributes F, to another set of *derived sequences* S', containing *derived objects* Q', and described by *derived attributes* F'.

$$T_{p_1, \ldots, p_k} : \langle S, Q, F \rangle \dashrightarrow \langle S', Q', F' \rangle$$

where $p_1, \ldots, p_k$ are parameters of the transformation that control its application. Each transformation may be applied iteratively, that is, the output of one transformation can be the input to a subsequent transformation. We have found four basic transformations to be especially useful for discovering sequence-generating rules: (a) *adding derived attributes*, (b) *segmenting*, (c) *splitting into phases*, and (d) *blocking*. Each of these is described in turn.

2.2.1 Adding derived attributes

The simplest transformation does not change the set of sequences, S, or the set of objects, Q, but only the set of attributes, F. For example, in Eleusis, the initial set F contains only two attributes: the RANK and SUIT of a card. These can be augmented by deriving such attributes as COLOR (red or black), FACEDNESS (faced or nonfaced), PARITY (odd or even), and PRIMENESS (prime or not prime in rank). Although the adding-derived-attributes transformation has no parameters, in cases where many such attributes could be derived, the program must use some heuristics to decide which attributes should be generated and added to the derived sequence.

2.2.2 Segmenting

The segmenting transformation derives a new sequence that is made up of a new set of objects, Q', which are described by a new set of attributes, F'. The new sequence is produced from the original sequence by dividing the original sequence into non-overlapping segments. Each segment becomes a derived object in the new sequence. The only parameter of this transformation is the segmentation condition that specifies how the original sequence is to be divided into segments. Three types of segmentation conditions are distinguished:

(a) those that use properties of the *original objects* to determine where the sequence *should* be broken, (b) those that use properties of the *original objects* to determine where the sequence should *not* be broken, and (c) those that use properties of *derived objects* to determine where the original sequence *should* be broken.

For example, suppose the original sequence consists of physical objects described by attributes such as WEIGHT, COLOR, and HEIGHT. An example of each type of segmentation condition follows:

1. Break when [WEIGHT(q_{i-1})>10][WEIGHT(q_i)$\leq$10].

   According to this condition, the original sequence is to be broken (between objects q_{i-1} and q_i) at the point where the weight of an object changes from above 10 to under 10.

2. Don't break as long as [COLOR(q_i)=COLOR(q_{i-1})][WEIGHT(q_i)>10].

   This condition states that the original sequence will not be broken (between q_{i-1} and q_i) if the color stays the same and the weight remains above 10. It will be broken at any point where either one of these conditions fails to hold.

3. Break so that [LENGTH(q_i')=2].

   This condition states that derived objects (q_i') should be subsequences of length 2 from the original sequence (i.e., pairs of adjacent objects from the original sequence).

The choice of attributes, F', for describing the newly-derived objects, Q', depends on the segmentation condition used to segment the sequence. For example, if the segmentation condition is [length(q_i')=2], attributes of interest might include the sum of the VALUES of the two original objects, the maximum VALUE, the minimum VALUE, and so on. The LENGTH of the segment would not be of interest, since by definition, it is a constant. However, if the segmentation condition is [color(q_i)=color(q_{i-1})], the LENGTH of the segment could be an interesting attribute and should be derived. Also, the COLOR shared by all of the cards in the segment might be of interest. In our implementation, the user provides an a priori knowledge base that specifies which attributes should be derived. Every user-specified attribute is derived unless the program can prove from the segmentation condition that the attribute would not have a well-defined value for each segment in the sequence or else would be trivially constant for all segments.

Often, a segmentation condition leads to the creation of incomplete segments at the beginning and end of the original sequence. These boundary cases can create difficulties during model instantiation, so they are ignored during rule discovery, but checked during rule evaluation.

2.2.3 Splitting

The splitting transformation splits a single sequence into a *sequence* of P separate subsequences called *phases*: $\langle ph_1, ph_2, \ldots \rangle$. Sequence ph_i starts with object q_i (the object at the i-th position in the original sequence) and continues with objects taken from succeeding positions at distance P apart in the original sequence. The objects in phase sequence ph_i are referred to as $\langle ph_{i,1}\ ph_{i,2}\ ph_{i,3} \ldots \rangle$. Hence, after splitting, derived object $ph_{i,j}$ is identical to original object $q_{i + P*(j-1)}$. P is the parameter of the splitting transformation and denotes the number of phases (the period length). Figure 5 shows the splitting operation with P = 3.

Original sequence: $\langle q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7\ q_8\ q_9 \rangle$
Derived sequence: $\langle ph_1\ ph_2\ ph_3 \rangle$, where

ph_1: $\langle ph_{1,1}$ $ph_{1,2}$ $ph_{1,3} \rangle$
ph_2: $\langle ph_{2,1}$ $ph_{2,2}$ $ph_{2,3} \rangle$
ph_3: $\langle ph_{3,1}$ $ph_{3,2}$ $ph_{3,3} \rangle$

(where $ph_{i,j} = q_k$ as indicated by vertical alignment, e.g., $ph_{1,2} = q_4$.)

Figure 5: Splitting transformation with P=3

The objects within each phase retain the linear ordering that they had in the original sequence. The phases themselves can be considered to be cyclically ordered so that ph_1 precedes ph_2, which precedes ph_3, and so on, until ph_P, which is followed by ph_1 again. Consider, for example, the following sequence:

$$\langle 1\ 8\ 2\ 9\ 3\ 10\ 4\ 11 \rangle$$

The splitting transformation with P=2 would produce the sequence $\langle ph_1\ ph_2 \rangle$ where

$$ph_1 = \langle 1\ 2\ 3\ 4 \rangle$$
$$ph_2 = \langle 8\ 9\ 10\ 11 \rangle$$

Since the splitting transformation simply breaks the original sequence of objects into subsequences of the same objects, no new descriptors are defined. The descriptors used to characterize objects in each of the phases are the same as those used to characterize the objects in the original sequence.

The splitting transformation can be applied to break one sequence-prediction problem into several subproblems—one for each phase. This enables the system to discover periodic rules.

2.2.4 Blocking

The blocking transformation converts the original sequence into a new sequence made up of a new set of objects B' and a new set of attributes F'. The new sequence is created by breaking the original sequence into overlapping segments called *blocks*. Each object b_i in the new sequence describes a block of L+1 consecutive

objects from the original sequence, starting at object q_i (called the *head*) and proceeding backwards to object q_{i-L} (where L is the *lookback* parameter of the blocking transformation). Figure 6 shows the blocking operation for L=2 (Block length of 3).

Original sequence: $\langle q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6 \quad q_7 \quad q_8 \rangle$
Derived sequence: $\langle b_3 \quad b_4 \quad b_5 \quad b_6 \quad b_7 \quad b_8 \rangle$
where b_i are derived objects defined as follows:

b_3:	$\langle q_1$	q_2	$\underline{q_3}\rangle$				
b4:		$\langle q_2$	q_3	$\underline{q_4}\rangle$			
b5:			$\langle q_3$	q_4	$\underline{q_5}\rangle$		
b6:				$\langle q_4$	q_5	$\underline{q_6}\rangle$	
b7:					$\langle q_5$	q_6	$\underline{q_7}\rangle$
b8:						$\langle q_6$	q_7 $\underline{q_8}\rangle$

The underlined objects are the head objects of each block.

Figure 6: The Blocking Transformation with Lookback Parameter L=2.

Several attributes are derived to describe each block. For each attribute Λ applicable to the objects in the original sequence, the attributes $\Lambda 0$, $\Lambda 1$, ..., ΛL are defined that are applicable to the objects in the derived sequence. $\Lambda 0(b_i)$ has the same value as $\Lambda(q_i)$; $\Lambda 1(b_i)$ has the same value as $\Lambda(q_{i-1})$; and so on until $\Lambda L(b_i)$, which has the same value as $\Lambda(q_{i-L})$. In other words, the original attributes are retained in the new sequence, but they are renamed so that they apply to whole *blocks* rather than to individual objects in the original sequence. The numerical suffix on the new names encodes the relative position of the original object q_i in block b_j.

For example, suppose the original sequence of objects is $\langle q_1\ q_2\ q_3\ q_4\ q_5 \rangle$ with attributes RANK and SUIT, where

q_1: [RANK(q_1)=2][SUIT(q_1)=H]

q_2: [RANK(q_2)=4][SUIT(q_2)=S]

q_3: [RANK(q_3)=6][SUIT(q_3)=C]

q_4: [RANK(q_4)=8][SUIT(q_4)=D]

q_5: [RANK(q_5)=10][SUIT(q_5)=H]

Suppose we apply the blocking transformation to this sequence with L=2 to obtain the derived sequence of blocks $\langle b_3\ b_4\ b_5 \rangle$. Then the descriptors RANK0, RANK1, RANK2, SUIT0, SUIT1, and SUIT2 will be derived with the values

b_3: [RANK2(b_3)=2][SUIT2(b_3)=H] &
[RANK1(b_3)=4][SUIT1(b_3)=S] &
[RANK0(b_3)=6][SUIT0(b_3)=C]

b_4: [RANK2(b_4)=4][SUIT2(b_4)=S] &
[RANK1(b_4)=6][SUIT1(b_4)=C] &
[RANK0(b_4)=8][SUIT0(b_4)=D]

b_5: [RANK2(b_5)=6][SUIT2(b_5)=C] &
[RANK1(b_5)=8][SUIT1(b_5)=D] &
[RANK0(b_5)=10][SUIT0(b_5)=H]

This transformation leads to a highly redundant representation of the information in the original sequence. For example, the information about SUIT and RANK of the original object q_3 is repeated as SUIT0 and RANK0 of block b_3, SUIT1 and RANK1 of block b_4, and SUIT2 and RANK2 of block b_5. However, this derived sequence of blocks facilitates the representation of the relationships between objects in the original sequence. Many sequence-prediction rules involve such relationships.

To represent relationships between objects, additional descriptors called *sum* and *difference* descriptors are defined. In the case of the above sequence, the descriptors S-RANK01, S-RANK02, D-RANK01, D-RANK02, D-SUIT01, and D-SUIT02 are created. The value of S-RANK01(b_i) is the sum of RANK0(b_i) and RANK1(b_i). The value of D-RANK01(b_i) is the difference between RANK0(b_i) and RANK1(b_i). Thus, in addition to the selectors shown above, the following selectors would also be derived for the new sequence:

b_3: [S-RANK01(b_3)=10][S-RANK02(b_3)=8] &
[D-RANK01(b_3)=2][D-RANK02(b_3)=4] &
[D-SUIT01(b_3)=1][D-SUIT02(b_3)=2]

b_4: [S-RANK01(b_4)=14][S-RANK02(b_4)=12] &
[D-RANK01(b_4)=2][D-RANK02(b_4)=4] &
[D-SUIT01(b_4)=1][D-SUIT02(b_4)=2]

b_5: [S-RANK01(b_5)=18][S-RANK02(b_5)=16] &
[D-RANK01(b_5)=2][D-RANK02(b_5)=4] &
[D-SUIT01(b_5)=1][D-SUIT02(b_5)=2]

Using this representation, it is relatively easy to discover that [D-RANK01(b_i)=2] is true for all blocks b_i.

Ordinarily, sum and difference attributes only make sense for attributes such as RANK whose domain sets are linearly ordered. We have extended the definition of difference to cover unordered and cyclically ordered

domain sets as well. For an unordered attribute such as COLOR, whose domain set is {red, black}, D-COLOR01 takes on the value 0 if the COLOR0(b_i) = COLOR1(b_i) and 1 otherwise. For attributes with cyclically-ordered domain sets, such as SUIT (with values {clubs, diamonds, hearts, spades}), D-SUIT01 is equal to the number of steps in the forward direction that are required to get from SUIT1(b_i) to SUIT0(b_i). If SUIT1(b_i) = diamonds and SUIT0(b_i) = clubs, D-SUIT01(b_i) = 3.

The sum and difference attributes make the ordering of the original sequence explicit in the attributes that describe each block. Consequently, it is no longer necessary to represent the ordering between blocks. Hence, the model-fitting algorithms discussed below treat the derived sequence (of blocks) as an unordered set of events.

One difficulty with the above notation is that the numerical suffixes are not very easy to read, especially when they are combined with sum or difference prefixes. Hence, we have developed an alternative representation that is more comprehensible. In this notation, selectors that refer to blocks, such as [SUIT1(b_i) = H], are written as selectors that refer to objects in the original sequence, such as [SUIT(q_{i-1}) = H]. Similarly, selectors such as [D-RANK01(b_i) = 3] are written as [RANK(q_i) = RANK(q_{i-1}) + 3]. This notation makes the meaning of the selectors clear without having to explicitly mention the blocks b_i.

For purposes of implementation, however, the first notation (which refers to blocks explicitly) is better because it enables the program to treat all sequences—including derived sequences—uniformly. In contrast to this, the second notation only works when at most one blocking transformation is applied. Multiple blocking transformations cannot be captured without explicitly mentioning the blocks. Because it is rare that more than one blocking transformation is needed and since the second notation is more understandable, we will use it for the rest of this chapter.

3 Representing Sequence-generating Rules and Models

A *sequence-generating rule* is a function g that assigns to each sequence of objects, $\langle q_1, q_2, \ldots, q_k \rangle$, a non-empty set of *admissible next objects* Q_{k+1}:

$$g(\langle q_1, q_2, \ldots, q_k \rangle) = Q_{k+1}$$

Q_{k+1} is the set of all objects that could appear as the next object in the sequence. For example, in the rule "Play a card whose rank is one higher than the previous card", the value of the function g when applied to a sequence whose last card is the 4C is the set of cards {5C, 5D, 5H, 5S}. This is written

$$g(\langle \ldots 4C \rangle) = Q_{k+1} = \{5C, 5D, 5H, 5S\}$$

Each set Q_{k+1} may contain only one event, or it may contain a large set of possible events. If for all k, the sequence $\langle q_1, q_2, ..., q_k \rangle$ is mapped by g into a singleton set, then the rule is a *deterministic* rule; otherwise, it is a *nondeterministic* rule. This chapter addresses the problem of discovering a nondeterministic sequence-generating rule, g, given the sequence $\langle q_1, q_2, ..., q_k \rangle$, where the q_i are characterized by a finite set of discrete attributes.

The sequence $\langle q_1, q_2, ..., q_k \rangle$ can be viewed as the set of assertions

$$q_1 \in g(\langle\rangle)$$
$$q_2 \in g(\langle q_1 \rangle)$$
$$\vdots$$
$$q_m \in g(\langle q_1, ..., q_{m-1} \rangle)$$

(Recall that the value of g(s), where s is a sequence, is a *set* of possible next objects.) These assertions are positive instances of the desired sequence-generating rule.

In Eleusis, negative instances are provided by the cards on the sidelines—that is, the cards indicated by the dealer as being incorrect continuations of the sequence. A sideline card q_4^- played after card q_3 provides a negative instance of the form:

$$q_4^- \notin g(\langle q_1, q_2, q_3 \rangle)$$

The goal is to find an expression for g that is *consistent* with these training instances and satisfies some preference criterion. (An expression g is consistent with the training instances if it characterizes all of the positive instances and excludes all of the negative instances.)

The preference criterion in our methodology, and generally in learning systems, attempts to evaluate a candidate rule in terms of its generality, predictive power, simplicity, and so on. These semantic properties are difficult to compute, however. Instead, virtually all learning systems employ syntactic criteria that correspond in some way to these semantic criteria. Syntactic criteria—such as the number of selectors in a conjunction and the number of conjuncts in a disjunction—will only correspond to the semantic criteria if the representational framework is well chosen (See McCarthy [1958]). As we noted in the introduction, most previous AI research on learning has employed a single representational framework or model for describing the rules or concepts to be learned. In Eleusis, a single framework is insufficient. Instead, we have developed three basic models that were found to be useful: *the DNF model*, *the decomposition model*, and *the periodic model*. When these models are employed, syntactic criteria can be used to approximate semantic criteria during evaluation.

A *model* is a structure that specifies a general syntactic form for a class of descriptions (in our case, sequence-generating rules). A model consists of *model parameters* and a set of *constraints* that the model places on the forms of descriptions. The process of specifying the values for the parameters of a model is called *parameterizing* the model. The process of filling in the form of the parameterized model is called *instantiating* the model. A fully-parameterized and fully-instantiated model forms a sequence-generating rule. Models can be instantiated using the original sequence, or, more typically, using a sequence derived by applying some of the data transformations discussed in the previous section.

All three models use the variable-valued logic calculus VL_{22} for representing sequence-generating rules. VL_{22} is an extension of the predicate calculus that uses the *selector* as its simplest kind of formula. The VL_{22} selector is substantially more expressive than the simple selector presented above in section 2.1. Recall that the simple selector has the form

$$[f_i(q_j)=r]$$

whereas the VL_{22} selector has the form

$$[f_i(x_1, x_2, ..., x_n) = r_1 \vee r_2 \vee ... \vee r_m].$$

In the VL_{22} selector, attributes f_i can take any number of arguments $(x_1, x_2, ..., x_n)$. Furthermore, the attributes f_i can take on any one of a *set* of values $\{r_1, r_2, ..., r_m\}$. The $\vee$ denotes the *internal disjunction* operator, that is, the disjunction of values of the same attribute. Thus, the selector

$$[\text{RANK}(q_i)=9 \vee 10 \vee J \vee Q \vee K]$$

indicates that the rank of object q_i can be either 9, 10, J, Q, or K. In this case, this same selector could be expressed alternatively as

$$[\text{RANK}(q_i)\geq 9],$$

since the domain of the RANK attribute is known to be linearly ordered with a maximum value of K (King). To aid comprehensibility, VL_{22} provides the operators $<$, $>$, $\leq$, $\geq$, and $\neq$, in addition to the basic $=$ operator.

Examples of typical selectors include:

- $[\text{RANK}(q_i)\neq\text{RANK}(q_{i-1})]$
  (paraphrase: the RANK of q_i is different from the RANK of q_{i-1})

- $[\text{SUIT}(q_i)=\text{SUIT}(q_{i-1})+1]$
  (paraphrase: the SUIT increases by one from q_{i-1} to q_i)

- $[\text{RANK}(q_i)+\text{RANK}(q_{i-2})>10]$
  (paraphrase: the sum of the RANKs of q_i and q_{i-2} is greater than 10)

Now that we have introduced the basic notation of VL_{22}, each of the three rule models is presented in turn.

3.1 The DNF Model

The DNF model supports the broad class of rules that can be expressed as a universally quantified VL_{22} statement in disjunctive normal form. The DNF model has one parameter, the degree of lookback, L. An example of a DNF rule (with L = 1) is

$$\forall i\ ([\text{COLOR}(q_i) = \text{COLOR}(q_{i-1})] \vee [\text{RANK}(q_i) = \text{RANK}(q_{i-1})])$$

which can be paraphrased as "Every object (q_i) in the sequence has the same color or the same rank (or both) as the preceding object (q_{i-1})."

In general, a DNF rule is a collection of conjuncts of the form

$$\forall i\ (C1(q_i) \vee C2(q_i) \vee C3(q_i) \vee \ldots \vee Ck(q_i))$$

The universal quantification over *i* indicates that this description is true for all objects q_i in the sequence.

An additional constraint specified in the DNF model is that the number of conjuncts, k, should be close to the minimum that produces a description consistent with the data.

3.2 The Decomposition Model

The decomposition model constrains the description to be a set of implications of the form:

$$L_1 \Rightarrow R_1$$
$$L_2 \Rightarrow R_2$$
$$\vdots$$
$$L_m \Rightarrow R_m$$

where the $\Rightarrow$ sign indicates logical implication.

The model states that the left- and right-hand sides, L_j and R_j, must all be VL_{22} conjunctions. The left-hand sides must be mutually exclusive and exhaustive—that is, the following two statements are true:

$$L_1 \vee L_2 \vee \ldots \vee L_m, \text{ and}$$

$$\forall j,k\ j \neq k \Rightarrow \neg\ (L_j \wedge L_k).$$

The first statement says that at least one of the left-hand sides L_j is always satisfied. The second statement says that if j and k are different, then L_j and L_k cannot both be satisfied simultaneously.

A decomposition rule describes the next object in the sequence in terms of characteristics of the previous objects in the sequence. For example, the rule

$$\forall i([\text{COLOR}(q_{i-1}) = \text{black}] \Rightarrow [\text{PARITY}(q_i) = \text{odd}]) \;\&\; ([\text{COLOR}(q_{i-1}) = \text{red}] \Rightarrow [\text{PARITY}(q_i) = \text{even}])$$

is a decomposition rule that says that if the last card was black, the next card must be odd, and if the last card was red, the next card must be even.

The decomposition model has a lookback parameter, L, that indicates how far back in the sequence the sequence-generating rule must "look" in order to predict the next object in the sequence. The above rule has a lookback parameter of 1, because is examines q_{i-1} (the previous object in the sequence).

3.3 The Periodic Model

This model consists of rules that describe objects in the sequence as having attribute values that repeat periodically. For example, the rule "Play alternating red and black cards" is a periodic rule. The periodic model has two parameters: the period length, P, and the lookback, L. The period length parameter, P, gives the number of phases in the periodic rule. A periodic rule can be viewed as applying a splitting transformation to split the original sequence into P separate sequences. Each separate phase sequence has a simple description. The lookback parameter, L, tells how far back, within a phase sequence, a periodic rule "looks" in order to predict the attributes of the next object in that phase. The periodic model imposes the additional constraint (or preference) that the different phases be disjoint (i.e., any given card is only playable within one phase).

A periodic rule is represented as an ordered P-tuple of VL_{22} conjunctions. The j-th conjunct describes the j-th phase sequence. The rule

$$\langle [\text{COLOR}(ph_{1,i}) = \text{red}], [\text{RANK}(ph_{2,i}) \geq \text{RANK}(ph_{2,i-1})] \rangle$$

is a periodic rule with P=2 and L=1, which says that the sequence is made of two (interleaved) phases. Each card in the first phase is red; each card in the second phase has at least as large a rank as the preceding card in that phase. Hence, one sequence that satisfies this rule is <2H 3C 10H 5S AD 6S 6H 6C>.

3.4 Derived Models

The three basic models can be combined to describe more complex rules. Basic models can be joined by conjunction, disjunction, and negation. For example, the rule "play alternating red and black cards such that the cards are in non-decreasing order" is a conjunction of the periodic rule

$$\langle [\text{COLOR}(ph_{1,i}) = \text{red}], [\text{COLOR}(ph_{2,i}) = \text{black}] \rangle$$

and the DNF rule

$$[\text{RANK}(q_i) \geq \text{RANK}(q_{i-1})].$$

3.5 Model Equivalences and the Heuristic Value of Models

The reader may have noticed that the decomposition and periodic models appear to be special cases of the DNF model. In particular, assuming that the left-hand side clauses in a decomposition rule are mutually-exclusive and exhaustive, the decomposition rule

$$L_1 \Rightarrow R_1$$
$$L_2 \Rightarrow R_2$$
$$\vdots$$
$$L_m \Rightarrow R_m$$

can be written as the DNF rule

$$[L_1 \,\&\, R_1] \vee [L_2 \,\&\, R_2] \vee \ldots \vee [L_m \,\&\, R_m]$$

Similarly, if the phases (C_i) of a periodic rule are mutually-exclusive and exhaustive, then the periodic rule

$$\langle C_1, C_2, \ldots, C_k \rangle$$

can be reexpressed as a decomposition rule of the form

$$C_1 \Rightarrow C_2$$
$$C_2 \Rightarrow C_3$$
$$\vdots$$
$$C_{k-1} \Rightarrow C_k$$
$$C_k \Rightarrow C_1.$$

This transformation from the periodic model into the decomposition model does not work when the phases of the periodic rule overlap. Consider, for example, the following periodic rule in which the phases are neither mutually exclusive nor exhaustive:

$$\langle [\text{COLOR}(ph_{1,i}) = \text{red}], [\text{RANK}(ph_{2,i}) = \text{even}] \rangle$$
(paraphrase: play alternating red and even cards)

Because the phases overlap, the transformation into a decomposition rule produces a slightly different rule:

$$[\text{COLOR}(q_{i-1}) = \text{red}] \Rightarrow [\text{PARITY}(q_i) = \text{even}] \,\&$$
$$[\text{PARITY}(q_{i-1}) = \text{even}] \Rightarrow [\text{COLOR}(q_i) = \text{red}]$$

To see how the two rules differ, consider the sequence of cards

<3D 2D 4C ...>

This sequence satisfies the second rule (the first *if-then* clause can be applied twice), but not the first rule (since the 4C is not red).

Even when the constraints of mutual exclusion and exhaustion are violated, it is always possible to develop some equivalent DNF rule for any periodic or decomposition rule. This is so because one can always provide additional descriptors that capture the particular relationship. The resulting DNF rules are not always as succinct or comprehensible, however, as the same rule expressed using one of the other models.

Consider this same periodic rule. Suppose that a new descriptor, called POSITION, is defined whose value for each object q_i is the position i of the object in the sequence. With this descriptor, the above rule can be encoded as

$$[\text{POSITION}(q_i)=\text{odd}] \Rightarrow [\text{COLOR}(q_i)=\text{red}] \;\&$$
$$[\text{POSITION}(q_i)=\text{even}] \Rightarrow [\text{PARITY}(q_i)=\text{even}]$$

Since any sequence-generating rule can be expressed in the DNF model, it is reasonable to ask why multiple models should be used at all. The answer is that the primary value of multiple models is that they provide heuristic guidance to the search for *plausible* rules. Hence, though the DNF model is capable of *representing* all of these rules, it is not helpful for *discovering* them. In short, it is *epistemologically adequate* but not *heuristically adequate* (see [McCarthy and Hayes, 1969; McCarthy, 1977]). Each model directs the attention of the learning system to a small subspace of the space of all possible DNF VL_{22} rules. The next section shows how the constraints associated with each model are incorporated into special model-fitting induction algorithms.

4 Architecture and Algorithms

Section 3 described the three basic processes involved in discovering sequence-generating rules: (a) transformation of the original sequence into a derived sequence, (b) selection of an appropriate model (and determination of its parameters) for the given sequence, and (c) fitting (instantiation) of the models to the derived sequence. Sections 4 and 5 presented the four data transformations and the three models. This section covers the third step of fitting specialized models to the transformed sequence. The model-fitting process is most easily understood in the context of the program architecture, so this section also discusses the system's architecture.

4.1 Overview of the System

The processes in the system (see Figure 7) are structured into four components—the three basic ones mentioned above plus an evaluation component. The processes of transforming the initial sequence and of selecting and parameterizing a model are performed in parallel. Then, model-fitting algorithms use the transformed sequence to instantiate the parameterized model to obtain a candidate sequence-generating rule. These candidate rules are then evaluated to determine the final set of rules.

The reason for performing data transformation and model selection in parallel is that these two processes are interdependent. For example, if a periodic model is selected (with period length P), then a splitting transformation (with number of phases P) needs to be applied to the sequence. These two processes can be viewed as simultaneous cooperative searches of two spaces: the space of possible data transformations and the space of possible parameterized models.

Figure 7: The Model-fitting Approach

4.2 Overview of the Concentric Ring Architecture

In order for the learning system to be easily modified to handle entire classes of NDP problems, the system is structured as a set of concentric knowledge rings (see Figure 8). A knowledge ring is a set of routines that perform a particular function using only knowledge appropriate to that function. The procedures within a given ring may invoke other procedures in that ring or in rings that are inside the given ring. Under these constraints, the concentric ring structure forms a hierarchically organized system.

Ideally, the rings should be organized so that the outermost ring uses the most problem-specific knowledge and performs the most problem-specific operations and the inner-most ring uses the most general knowledge and performs the most general tasks. Such an architecture improves the program's generality because it can

Figure 8: The knowledge ring architecture

be applied to increasingly different NDP problems by removing and replacing the outer rings. In order to apply the program to radically different learning problems, all but the inner-most ring may need to be replaced.

The ring architecture is used here as follows. The outer-most rings perform user-interface functions and convert the initial sequence from whatever domain-specific notation is being used into a sequence of VL_{22} events. The inner-most ring performs the model-fitting functions. It expects the data to be properly transformed so that the data have the same form as the models to which they are to be fitted. The intervening rings conduct the simultaneous processes of developing a properly parameterized model and transforming the input sequence into an appropriate derived sequence.

The intervening rings also evaluate the rules discovered by the inner-most ring using the knowledge available in each ring.

4.3 The System SPARC/E

SPARC (Sequential PAttern ReCognition) stands for a general program designed to solve a variety of nondeterministic prediction problems using the ring architecture. So far, we have implemented only a more specific version of the program, called SPARC/E, tailored specifically to the problem of rule discovery in the game Eleusis. SPARC/E is made up of five rings, as shown in Figure 8.

This section describes the functions of each ring in SPARC/E. To illustrate these ring functions, we use the Eleusis layout shown in Figure 9. Recall that in an Eleusis layout, the main line shows the correctly-played sequence of cards (positive examples). The side lines, which branch out below the main line, contain cards that do not satisfy the rule—that is, incorrect continuations of the sequence (negative examples).

```
Main line:  3H  9S  4C  JD  2C  10D  8H  7H  2C
Side lines:     JD      AH  AS          10H
                5D      8H  10S
                        QD
```

Figure 9: Sample Eleusis Layout

4.3.1 Ring 5: User interface

Ring 5, the outer-most ring, provides a user interface to the program. It executes user's commands for playing the card game Eleusis, as well as commands for controlling the search, data transformation, generalization, and evaluation functions of the program. One command in Ring 5 is the INDUCE command that instructs SPARC/E to look for plausible NDP rules that describe the current sequence. When the INDUCE command is given, Ring 5 calls Ring 4 to begin the rule discovery process. Ring 5 provides Ring 4 with an initial sequence of VL_{22} events in which the only attributes are SUIT and RANK.

4.3.2 Ring 4: Adding derived attributes

Ring 4 applies the adding-derived-attributes transformation to the initial sequence of cards. This involves creating derived attributes that make explicit certain commonly known characteristics of playing cards that are likely to be used in an Eleusis rule: COLOR, PARITY, FACED versus NON-FACED cards, and so on. The user of the program provides a definition for each descriptor that is to be derived. Figure 10 shows the layout from Figure 9 after it has been processed by Ring 4. The plusses and minuses along the right-hand side of the figure indicate whether the event is a positive example or a negative example of the sequence-generating rule. These derived events are passed to Ring 3 for further processing.

4.3.3 Ring 3: Segmenting the layout

Ring 3 is the first Eleusis-independent ring. It applies the segmenting transformation to the sequence supplied by Ring 4. In the present implementation, the end points of each segment are determined by applying a segmentation predicate, $P(card_{i-1}, card_i)$ to all pairs of adjacent events in the sequence. When the predicate P evaluates to FALSE, the sequence is broken between $card_{i-1}$ and $card_i$ to form the end of a segment. Typical segmentation predicates used are:

$$[rank(card_i) = rank(card_{i-1})]$$
$$[rank(card_i) = rank(card_{i-1}) + 1]$$
$$[color(card_i) = color(card_{i-1})]$$
$$[suit(card_i) = suit(card_{i-1})]$$
$$[parity(card_i) = parity(card_{i-1})]$$

VL_{22} event	Positive or negative
[RANK($card_1$) = 3][SUIT($card_1$) = H] & [PARITY($card_1$) = odd][COLOR($card_1$) = red] & [PRIME($card_1$) = N][FACED($card_1$) = Y]	+
[RANK($card_2$) = 9][SUIT($card_2$) = S] & [PARITY($card_2$) = odd][COLOR($card_2$) = black] & [PRIME($card_2$) = N][FACED($card_2$) = N]	+
[RANK($card_3$) = J][SUIT($card_3$) = D] & [PARITY($card_3$) = odd][COLOR($card_3$) = red] & [PRIME($card_3$) = Y][FACED($card_3$) = Y]	-
[RANK($card_3$) = 5][SUIT($card_3$) = D] & [PARITY($card_3$) = odd][COLOR($card_3$) = red] & [PRIME($card_3$) = N][FACED($card_3$) = Y]	-
[RANK($card_3$) = 4][SUIT($card_3$) = C] & [PARITY($card_3$) = even][COLOR($card_3$) = black] & [PRIME($card_3$) = N][FACED($card_3$) = N]	+
[RANK($card_4$) = J][SUIT($card_4$) = D] & [PARITY($card_4$) = odd][COLOR($card_4$) = red] & [PRIME($card_4$) = Y][FACED($card_4$) = Y]	+
etc.	

Figure 10: Derived layout after Ring 4 processing of the layout in Figure 9.

Other techniques for performing segmentation, such as providing a predicate that becomes TRUE at a segment boundary (see section 2.2.2), are not implemented in SPARC/E.

Ring 3 searches the space of possible segmentations using two search pruning heuristics. After each attempt to segment the sequence, it counts the number of derived objects (segments), k, in the derived sequence. If k is less than 3, the segmentation is discarded since there are too few derived objects to use for generalization. If k is more than half of the number of objects in the original sequence, the segmentation is also discarded because in this case many segments contain only one original object. Segmented sequences that survive these two pruning heuristics are passed on to Ring 2 for further processing.

One segmentation that Ring 3 always performs is the "null" segmentation—that is, it always passes the unsegmented sequence directly to the inner rings. Figure 11 shows a sample layout and the resulting derived layout after segmentation using the segmentation condition: [SUIT(card$_i$)=SUIT(card$_{i-1}$)]. The derived objects (segments) are denoted by variables segment$_i$. The negative event [SUIT(segment$_2$)=D] [COLOR(segment$_2$)=red] [LENGTH(segment$_2$)=3] is obtained from the segment **<5D 2D 4D>**, which ends in a "side-line" card. Notice that the very last card in the sequence, the king of spades, is not included in any segment. This is because the king is the first card of a new segment, and it is impossible to know how long that segment will be until it is completed. Once a sequence-generating rule is found by the inner rings, Ring 3 will check to make sure that the king of spades is consistent with the rule.

The layout:

3H	6D	2D	7C	AC	9C	JH	6H	8H	QH	KS
	6S	4D		AH						
				7S						

The derived sequence:

Description of derived object	Positive or negative
[SUIT(segment$_1$)=H][COLOR(segment$_1$)=red] & [LENGTH(segment$_1$)=1]	+
[SUIT(segment$_2$)=D][COLOR(segment$_2$)=red] & [LENGTH(segment$_2$)=2]	+
[SUIT(segment$_2$)=D][COLOR(segment$_2$)=red] & [LENGTH(segment$_2$)=3]	-
[SUIT(segment$_3$)=C][COLOR(segment$_3$)=black] & [LENGTH(segment$_3$)=3]	+
[SUIT(segment$_4$)=H][COLOR(segment$_4$)=red] & [LENGTH(segment$_4$)=4]	+

Figure 11: Sample layout and segmented sequence under segmentation condition [SUIT(card$_i$)=SUIT(card$_{i-1}$)]

SPARC/E derives the descriptors COLOR, SUIT, and LENGTH to describe each derived object. The choice of which descriptors to derive involves three steps. First, LENGTH is derived whenever the segmentation transformation is applied. Second, any descriptor that is tested in the segmentation predicate (in this case, SUIT) is also derived. Third, any descriptor is derived whose value can be proved to be the same for all cards in each segment. In this case, COLOR is derived because, if SUIT is a constant, then COLOR is also a constant. Using this segmentation, SPARC can use the DNF model to discover that the segmented sequence can be described as

$$[\text{LENGTH}(\text{segment}_i) = \text{LENGTH}(\text{segment}_{i-1}) + 1]$$

That is, the LENGTH of each segment of constant SUIT (in the main line) increases by 1.

4.3.4 Ring 2: Parameterizing the models

Ring 2 searches the space of parameterizations of the three basic models. Each model is considered in turn. For each model, Ring 2 develops a set of derived events based on each allowed value of the lookback parameter, L, and the number of phases parameter, P. The user can control which models should be inspected and what range of values for L and P should be investigated. By default, the program will inspect the decomposition model with L = 0, 1, or 2, and the periodic model with P = 1 or 2 and L = 0 or 1. The DNF model is not inspected under the default settings for the program.

Specifically, Ring 2 performs the following actions depending on which model is being parameterized:

A. For the *decomposition* model with lookback parameter L, Ring 2 applies the blocking transformation to break the sequence received from Ring 3 into blocks of length L. After blocking, all of the attributes that described the original objects are converted into attributes that describe the whole block (as discussed in section 4 above). Furthermore, sum and difference descriptors are derived to represent the relationships between adjacent objects in the original sequence. The resulting derived events can be viewed as very specific if-then clauses of the following form.

Given an initial sequence of objects $\langle q_1, q_2, \ldots, q_m \rangle$, let us look at block b_i which describes the subsequence $\langle q_{i-L}, \ldots, q_{i-1}, q_i \rangle$. Let F_j, $j = 0, 1, \ldots, L$, denote the set of selectors describing object q_{i-j} renamed so that they apply to block b_i. For example, F_1 could be the selectors $[\text{SUIT1}(b_i) = H][\text{RANK1}(b_i) = 3]$—selectors that originally referred to object q_{i-1}. Let $d(F_j, F_k)$ denote all of the difference selectors obtained by "subtracting" event F_k from event F_j, and let $s(F_j, F_k)$ denote all of the summation selectors obtained by "summing" events F_j and F_k. For example, $d(F_0, F_1)$ could include the selectors $[\text{D-SUIT01}(b_i) = 2][\text{D-RANK01}(b_i) = -3]$ obtained from "subtracting" F_1 from F_0.

With these definitions, the derived events for the decomposition model have the form:

$$F_1 \& \ldots \& F_L \Rightarrow F_0 \& d(F_0, F_1) \& \ldots \& d(F_0, F_L) \& s(F_0, F_1) \& \ldots \& s(F_0, F_L)$$

Suppose, for example, that the initial sequence of cards is

<2H 4D 6S 8C>

with only the SUIT and RANK descriptors being employed. Then suppose that Ring 2 applies the blocking transformation with a lookback of 2. Figure 12 shows the two derived events that will be produced by Ring 2 (the corresponding notation is shown to the right of each group of selectors).

Derived event	Abbreviation
$[\text{RANK1}(q_{i-1})=4][\text{SUIT1}(q_{i-1})=\text{D}]$	F_1
$[\text{RANK2}(q_{i-2})=2][\text{SUIT2}(q_{i-2})=\text{H}]$	F_2
$\Rightarrow$	
$[\text{RANK0}(q_i)=6][\text{SUIT0}(q_i)=\text{S}]$	F_0
$[\text{D-RANK01}(q_i,q_{i-1})=2][\text{D-SUIT}(q_i,q_{i-1})=2]$	$d(F_0,F_1)$
$[\text{D-RANK02}(q_i,q_{i-2})=4][\text{D-SUIT}(q_i,q_{i-2})=1]$	$d(F_0,F_2)$
$[\text{S-RANK01}(q_i,q_{i-1})=10]$	$s(F_0,F_1)$
$[\text{S-RANK02}(q_i,q_{i-2})=8]$	$s(F_0,F_2)$
$[\text{RANK1}(q_{i-1})=6][\text{SUIT1}(q_{i-1})=\text{S}]$	F_1
$[\text{RANK2}(q_{i-2})=4][\text{SUIT2}(q_{i-2})=\text{D}]$	F_2
$\Rightarrow$	
$[\text{RANK0}(q_i)=8][\text{SUIT0}(q_i)=\text{C}]$	F_0
$[\text{D-RANK01}(q_i,q_{i-1})=2][\text{D-SUIT}(q_i,q_{i-1})=1]$	$d(F_0,F_1)$
$[\text{D-RANK02}(q_i,q_{i-2})=4][\text{D-SUIT}(q_i,q_{i-2})=3]$	$d(F_0,F_2)$
$[\text{S-RANK01}(q_i,q_{i-1})=14]$	$s(F_0,F_1)$
$[\text{S-RANK02}(q_i,q_{i-2})=12]$	$s(F_0,F_2)$

Figure 12: Sample events showing which sum and difference descriptors are derived.

These derived events no longer need to be ordered, since the ordering information is made explicit within the events. These events have the form of very specific if-then clauses. This facilitates the model-fitting process in Ring 1.

B. For the *DNF* model with lookback parameter L, the sequence derived in Ring 3 is blocked in a very similar manner, except that only the selectors describing q_i are retained in the description of block b_i. The derived events have the following form:

$$F_0 \,\&\, d(F_0, F_1) \,\&\, \ldots \,\&\, d(F_0, F_L) \,\&\, s(F_0, F_1) \,\&\, \ldots\, s(F_0, F_L)$$

These events are very specific conjuncts that are passed to the Λ^q algorithm [Michalski, 1969, 1972] in Ring 1, where they are generalized to form a DNF description.

C. For the *periodic model* with period length P and lookback L, Ring 2 performs a splitting transformation followed by a blocking transformation. First, the sequence obtained from Ring 3 is split into P separate sequences. Then each separate sequence is blocked into blocks of length L+1. The derived events have the same form as the events derived for the DNF model. Note that because the blocking occurs after the splitting, the lookback takes place only within a phase.

To provide an example of the function of Ring 2, Figure 13 shows some events from Figure 10 after they have been transformed in preparation for fitting to a decomposition model with L=1.

4.3.5 Ring 1: The basic model-fitting algorithms

Ring 1 consists of three separate model-fitting algorithms: the Λ^q algorithm, the decomposition algorithm and the periodic algorithm.

The Λ^q algorithm [Michalski, 1969, 1972] is applied to fit the DNF model to the data. Λ^q attempts to find the DNF description with the fewest number of conjunctive terms that covers all of the positive examples and none of the negative examples. The algorithm operates as follows. First, a positive example, called the seed, is chosen, and the set of maximally-general conjunctive expressions consistent with this seed and all of the negative examples is computed. This set is called a *star*, and it is equivalent to the G-set in Mitchell's [1978] version space approach (if the G-set is computed with the seed positive example and all of the negative examples). One element from this star is chosen to be a conjunct in the output DNF description, and all positive examples covered by it are removed from further consideration. If any positive examples remain, the process is repeated, selecting as a new seed some positive example that was not covered by *any* member of any preceding star. In this manner, a DNF description with few conjunctive terms is found. If the stars are computed without any pruning, then Λ^q can provide a tight bound on the number of conjuncts that would appear in the shortest DNF description (i.e., with fewest conjunctive terms).

The decomposition algorithm is an iterative algorithm that seeks to fit the data to a decomposition model. The key task of the decomposition algorithm is to identify a few attributes, called *decomposition attributes*, from which the decomposition rule can be developed. A *decomposition attribute* is an attribute that appears on the left-hand side of an if-then clause of a decomposition rule. For example, the decomposition rule

Derived event	Positive or negative
$[\text{RANK1}(b_2)=3][\text{SUIT1}(b_2)=\text{H}]$ $[\text{PARITY1}(b_2)=\text{odd}][\text{COLOR1}(b_2)=\text{red}]$ $[\text{PRIME1}(b_2)=\text{Y}][\text{FACED1}(b_2)=\text{N}]$	
$\Rightarrow$	
$[\text{RANK0}(b_2)=9][\text{SUIT0}(b_2)=\text{S}][\text{PARITY0}(b_2)=\text{odd}]$ $[\text{COLOR0}(b_2)=\text{black}][\text{PRIME0}(b_2)=\text{N}]$ $[\text{FACED0}(b_2)=\text{N}][\text{D-RANK01}(b_2)=+6]$ $[\text{D-SUIT01}(b_2)=+1][\text{D-PARITY01}(b_2)=\text{N}]$ $[\text{D-COLOR01}(b_2)=\text{Y}][\text{D-PRIME01}(b_2)=\text{Y}]$ $[\text{D-FACED01}(b_2)=\text{Y}][\text{S-RANK01}(b_2)=12]$	+
$[\text{RANK1}(b_3)=9][\text{SUIT1}(b_3)=\text{S}]$ $[\text{PARITY1}(b_3)=\text{odd}][\text{COLOR1}(b_3)=\text{black}]$ $[\text{PRIME1}(b_3)=\text{N}][\text{FACED1}(b_3)=\text{N}]$	
$\Rightarrow$	
$[\text{RANK0}(b_3)=\text{J}][\text{SUIT0}(b_3)=\text{D}][\text{PARITY0}(b_3)=\text{odd}]$ $[\text{COLOR0}(b_3)=\text{red}][\text{PRIME0}(b_3)=\text{Y}]$ $[\text{FACED0}(b_3)=\text{Y}][\text{D-RANK01}(b_3)=+2]$ $[\text{D-SUIT01}(b_3)=+2][\text{D-PARITY01}(b_3)=\text{N}]$ $[\text{D-COLOR01}(b_3)=\text{Y}][\text{D-PRIME01}(b_3)=\text{Y}]$ $[\text{D-FACED01}(b_3)=\text{Y}][\text{S-RANK01}(b_3)=20]$	-

Figure 13: Some events of Figure 10 transformed for decomposition L=1.

$[\text{COLOR}(\text{card}_{i-1})=\text{black}] \Rightarrow [\text{PARITY}(\text{card}_i)=\text{odd}]$ &
$[\text{COLOR}(\text{card}_{i-1})=\text{red}] \Rightarrow [\text{PARITY}(\text{card}_i)=\text{even}]$

decomposes on COLOR. Hence, COLOR is the single decomposition attribute.

The algorithm uses a generate-and-test approach of the following form:

```
decomposition-attributes := {}  The empty set

while rule is not consistent do
    begin

    generate a trial decomposition
    (based on positive evidence only)
    for each possible decomposition attribute

    test these trial decompositions against
    the data

    select the best decomposition attribute and
    add it to the set decomposition-attributes

    end
```

The process of generating a trial decomposition takes place in two steps. First, a VL_{22} conjunction is formed for each possible value of the decomposition attribute. All positive events that have the same value of the decomposition attribute on their left-hand sides are merged together to form a single conjunction of selectors. This VL_{22} conjunction forms the right-hand side of a single clause in the decomposition rule. Within this conjunction, a selector is created for each attribute by forming the internal disjunction of the values in the corresponding selectors in the events. For example, using all of the events derived in Ring 2 for the sample layout in Figure 9, the decomposition algorithm generates the trial decomposition shown in Figure 14 for the PARITY($card_{i-1}$) attribute.

Since there are only two values (ODD and EVEN) for the decomposition attribute in the sequence shown in Figure 9, two conjunctions are formed. The first conjunction is obtained by merging all of the positive events for which [PARITY($card_{i-1}$)=odd]. There are four such events. The first selector in that conjunction, [RANK($card_i$)=9 ∨ 4 ∨ 2], is obtained by forming the internal disjunction of the values of RANK($card_i$) in each of the four events.

The second step in forming a trial decomposition is to generalize each clause in the trial rule. The generalization is accomplished by applying rules of generalization to extend internal disjunctions and drop selectors. (See [Michalski, 1983] for a description of various rules of generalization.) Corresponding attributes in the different clauses of the decomposition rule are compared, and selectors whose value sets overlap are dropped. When these rules of generalization are applied to the trial decomposition of, for example, PARITY, the following generalized trial decomposition is obtained:

[PARITY($card_{i-1}$)=odd] ⇒ [SUIT($card_i$)=C ∨ S][COLOR($card_i$)=black] &
[PARITY($card_{i-1}$)=even] ⇒ [SUIT($card_i$)=H ∨ D][COLOR($card_i$)=red]

This is a very promising trial decomposition. However, it has been developed using only positive evidence—without considering the possibility that it may cover some negative events. Hence, the trial

[PARITY($card_{i-1}$) = odd]

⇒

[RANK($card_i$)) = 9 v 4 v 2]
[SUIT($card_i$) = S v C][PARITY($card_i$) = even v odd]
[COLOR($card_i$) = black][PRIME($card_i$) = Y v N]
[FACED($card_i$) = N]
[D-RANK($card_i$,$card_{i-1}$) = +6 v -5 v -7]
[D-SUIT($card_i$,$card_{i-1}$) = 1 v 2 v 3]
[D-PARITY($card_i$,$card_{i-1}$) = Y v N]
[D-COLOR($card_i$,$card_{i-1}$) = Y v N]
[D-PRIME($card_i$,$card_{i-1}$) = Y v N]
[D-FACED($card_i$,$card_{i-1}$) = Y v N]
[S-RANK($card_i$,$card_{i-1}$) = 12 v 13 v 9] &

[PARITY($card_{i-1}$) = even]

⇒

[RANK($card_i$) = J v 10 v 8 v 7]
[SUIT($card_i$) = H v D][PARITY($card_i$) = even v odd]
[COLOR($card_i$) = red][PRIME($card_i$) = Y v N]
[FACED($card_i$) = Y v N]
[D-RANK($card_i$,$card_{i-1}$) = 7 v 8 v -2 v -1]
[D-SUIT($card_i$,$card_{i-1}$) = 0 v 1]
[D-PARITY($card_i$,$card_{i-1}$) = Y v N]
[D-COLOR($card_i$,$card_{i-1}$) = Y v N]
[D-PRIME($card_i$,$card_{i-1}$) = Y v N]
[D-FACED($card_i$,$card_{i-1}$) = Y v N]
[S-RANK($card_i$,$card_{i-1}$) = 15 v 12 v 18]

Figure 14: Trial decomposition on the PARITY($card_{i-1}$) attribute

decomposition must be tested against the negative events to determine whether or not it is consistent. It turns out that the generalized trial decomposition shown above is indeed consistent with the negative evidence.

After a trial decomposition has been developed for each possible decomposition attribute, the best decomposition attribute is selected according to a heuristic attribute-quality functional. The attribute-quality

functional tests such things as the number of negative events covered by the trial decomposition, the number of clauses with non-null right-hand sides, and the complexity of the trial decomposition (defined as the number of selectors that cannot be written with a single operator and a single value). The chosen trial decomposition forms a candidate sequence-prediction rule.

If the candidate rule is not consistent with the data (i.e., still covers some negative examples), then the decomposition algorithm must be repeated to select a second attribute to add to the left-hand sides of the if-then clauses. This has the effect of splitting each of the if-then clauses into several more if-then clauses. For example, if we first decomposed on PARITY($card_{i-1}$) and then on FACED($card_{i-1}$), we would obtain four if-then clauses of the form:

[PARITY($card_{i-1}$)=odd][FACED($card_{i-1}$)=N] ⇒ ...
[PARITY($card_{i-1}$)=odd][FACED($card_{i-1}$)=Y] ⇒ ...
[PARITY($card_{i-1}$)=even][FACED($card_{i-1}$)=N] ⇒ ...
[PARITY($card_{i-1}$)=even][FACED($card_{i-1}$)=Y] ⇒ ...

The periodic algorithm is similar to the decomposition algorithm. For each phase of the period, it takes all of the positive events in that phase and combines them to form a single conjunct by forming the internal disjunction of all of the value sets of corresponding selectors. Next, rules of generalization are applied to extend internal disjunctions and drop selectors. Finally, corresponding attributes in different phases are compared, and selectors whose value sets overlap are dropped if this can be done without covering any negative examples.

4.3.6 Evaluating the NDP rules

Once Ring 1 has instantiated the parameterized models to produce a set of rules, the rules are passed back through the concentric rings of the program. Each ring evaluates the rules according to plausibility criteria based on knowledge available in that ring. Ring 2, for example, applies knowledge of the fact that valid sequences can be continued indefinitely. It checks to see that the rule predicts that the sequence could be so-continued. Ring 3—which applies the segmentation transformation—applies its knowledge about tail end of the unsegmented sequence to make sure it is consistent with the rule. (Recall that the segmentation transformation is sometimes unable to segment the last few events in the sequence.) Ring 4 tests the rule using the plausibility criteria for Eleusis. These criteria are:

1. Prefer rules with intermediate degree of complexity. In Eleusis, Occam's Razor does not always apply. The dealer is unlikely to choose a rule that is extremely simple, because it would be too easy to discover. Very complex rules will not be discovered by anyone, and, since the rules of the game discourage such an outcome, the dealer is not likely to choose such complex rules either.

2. Prefer rules with an intermediate degree of non-determinism. Rules with a low degree of non-determinism lead to many incorrect plays, thus rendering them easy to discover. Rules that are very nondeterministic generally lead to few incorrect plays and are therefore difficult to discover.

Rules that do not satisfy these heuristic criteria are discarded. The remaining rules are returned to Ring 5 where they are printed for the user.

5 Examples of Program Execution

In this section, we present some example Eleusis games and the corresponding sequence-generating laws that were discovered by SPARC/E. Each of these games was an actual game among people, and the rules are presented as they were displayed by SPARC/E (with minor typesetting changes).

The raw sequences presented to SPARC/E had only two attributes: SUIT and RANK. SPARC/E was given definitions of the following derivable attributes:

- COLOR (red for Hearts and Diamonds; black for Clubs and Spades)
- FACE (true if card is a faced, picture card, false otherwise)
- PRIME (true if card has a prime rank, false otherwise)
- MOD2 (the parity value of the card, 0 if card is even, 1 otherwise)
- MOD3 (the rank of the card modulo 3)
- LENMOD2 (when SPARC/E segments the main sequence into derived subsequences, it computes the LENGTH of each of the subsequences modulo 2)

Three examples of the program execution are presented. Here are some points to notice in reading the examples. First, each rule is assumed to be universally quantified over all events in the sequence. This quantification is not explicitly printed. Second, when the value set of a selector includes a set of adjacent values (e.g., $[\text{RANK}(\text{card}_i)=3 \vee 4 \vee 5]$, this is printed as `[RANK(CARDI)=3..5]`. The computation times given are for an implementation in PASCAL on the CDC CYBER 175.

5.1 Example 1

In this example, we show the program discovering a segmented rule. The program was presented with the following layout:

```
Main line:    AH  7C  6C  9S  10H 7H  10D JC  AD
Side lines:       KD      6S      QD
                  JH

continued:    4H  8D  7C  9S  10C KS  2C  10S JS
              3S          9H      QH
                          6H      AD
```

The program only discovered one rule for this layout, precisely the rule that the dealer had in mind (1.2 seconds required):

```
RULE 1: LOOKBACK: 0 NPHASES: 1 PERIODIC MODEL

SEGMENTATION CONDITION = [COLOR(CARDI)=COLOR(CARDI-1)]:

                         PERIOD< [LENMOD2(PH1SEGMENTI)=1] >
```

The rule states that one must play strings of cards with the same color. The strings must always have odd length. The segmentation condition states that a segment is a string of cards all of the same color. CARDI refers to the I-th card in the original sequence. SPARC/E discovered this rule as a degenerate periodic rule with a period length, P, of 1. Hence, PH1SEGMENTI refers to the I-th segment in phase 1 (the only phase) of the derived sequence. Actually, the rule that the dealer had in mind had one additional constraint: a queen must not be played adjacent to a jack or king.

5.2 Example 2

The second example requires the program to discover a fairly simple periodic rule. Here is the layout:

```
Main line:    JC  4D  QH  3S  QD  9H  QC  7H  QD
Side lines:   KC  6S              4S      10D
                  7S

Main line continued:    9D  QC  3H  KH  4C  KD  6C  JD  8D

Main line continued:    JH  7C  JD  7H  JH  6H  KD
```

The program discovered three equivalent versions of the rule, which can be paraphrased as "Play alternating faced and non-faced cards." Here are the rules (0.49 seconds were required):

```
RULE 1: LOOKBACK: 1 NPHASES: 0 DECOMPOSITION MODEL

[FACE(CARDI-1)=FALSE] =>
   [RANK(CARDI)≥JACK]
   [RANK(CARDI)>RANK(CARDI-1)]
   [FACE(CARDI)=TRUE]            &

[FACE(CARDI-1)=TRUE] =>
   [RANK(CARDI)=3..9]
   [RANK(CARDI)<RANK(CARDI-1)]
   [FACE(CARDI) =FALSE]


RULE 2: LOOKBACK: 1 NPHASES: 1 PERIODIC MODEL

PERIOD<[RANK(PH1CARDI)≥3]
       [RANK(PH1CARDI)≠RANK(PH1CARDI-1)]
       [FACE(PH1CARDI)≠FACE(PH1CARDI-1)]>


RULE 3: LOOKBACK: 1 NPHASES: 2 PERIODIC MODEL

PERIOD<[RANK(PH1CARDI)≥JACK]
       [RANK(PH1CARDI)≥-RANK(PH1CARDI-1)+20]
       [FACE(PH1CARDI)=TRUE],

       [RANK(PH2CARDI)=3..9]
       [RANK(PH2CARDI)=-RANK(PH2CARDI-1)+5..14]
       [FACE(PH2CARDI)=FALSE]>
```

Rule 1 is a decomposition rule with a lookback of 1. Rule 2 expresses the rule as a degenerate periodic rule with a single phase. Rule 3 expresses the rule in the "natural" way as a periodic rule of length 2.

Notice that, although the program has the gist of the rule, it has discovered a number of redundant conditions. For example, in rule 1, the program did not use knowledge of the fact that [RANK($card_i$)≥jack] implies [FACE($card_i$)=true], and therefore, it did not remove the former selector. Similarly, because of the interaction of the two conditions in rule 1, [RANK($card_i$)>RANK($card_{i-1}$)] is completely redundant.

5.3 Example 3

The third example shows the upper limits of the program's abilities. During this game, only one of the human players even got close to guessing the rule, yet the program discovers a good approximation of the rule using only a portion of the layout that was available to the human players. Here is the layout:

```
Main line:   4H  6D  8C  JS  2C  6S  AC  6S  10H
Side lines:  7C  6S  KC  AH      6C      AS
             JH  7H  3H  KD
             4C  2C      QS
             10S 7S
             8H  6D
             AD  6H
             2D  4C
```

The program produced the following rules after 6.5 seconds:

```
RULE 1: LOOKBACK: 1 NPHASES: 0 DNF MODEL

[RANK(CARDI)≤5][SUIT(CARDI)=SUIT(CARDI-1)+1] V
[RANK(CARDI)≥6][SUIT(CARDI)=SUIT(CARDI-1)+3]


RULE 2: LOOKBACK: 1 NPHASES: 1 PERIODIC MODEL

PERIOD<[RANK(PH1CARDI)=RANK(PH1CARDI-1)-9]
   [RANK(PH1CARDI)=-RANK(PH1CARDI-1)+4,5,7,11,13,17]
   [SUIT(PH1CARDI)=SUIT(PH1CARDI-1)+1,2,3]>


RULE 3: LOOKBACK: 1 NPHASES: 2 PERIODIC MODEL

PERIOD<[RANK(PH1CARDI)=ACE,2,8,10]
   [RANK(PH1CARDI)=-RANK(PH1CARDI-1)+1,8,9,10],

   [RANK(PH2CARDI)=6..JACK][SUIT(PH2CARDI)=SPADES]
   [RANK(PH2CARDI)=RANK(PH2CARDI-1)+-0..6]
   [RANK(PH2CARDI)=-RANK(PH2CARDI-1)+8..14]
   [SUIT(PH2CARDI)=SUIT(PH2CARDI-1)+0..2]
   [COLOR(PH2CARDI)=BLACK][PRIME(PH2CARDI)=PTRUE]
   [PRIME(PH2CARDI)=PRIME(PH2CARDI-1)]
   [MOD2(PH2CARDI)=1][MOD2(PH2CARDI)=MOD2(PH2CARDI-1)+0]
   [MOD2(PH2CARDI)=-MOD2(PH2CARDI-1)+0][MOD3(PH2CARDI)=2]
   [MOD3(PH2CARDI)=MOD3(PH2CARDI-1)+0]
   [MOD3(PH2CARDI)=-MOD3(PH2CARDI-1)+1]>
```

The rule that the dealer had in mind was:

$[\text{SUIT}(\text{card}_i) = \text{SUIT}(\text{card}_{i-1}) + 3]$
$[\text{RANK}(\text{card}_i) \geq \text{RANK}(\text{card}_{i-1})]$ V

$[\text{SUIT}(\text{card}_i) = \text{SUIT}(\text{card}_{i-1}) + 1]$
$[\text{RANK}(\text{card}_i) \leq \text{RANK}(\text{card}_{i-1})]$

There is a strong symmetry in this rule: the players may either play a lower card in the next "higher" suit (recall that the suits are cyclically ordered) or a higher card in the next "lower" suit. The program discovered a slightly simpler version of the rule (rule 1) that happened to be consistent with the training instances. Note that adding 3 to the SUIT has the effect of computing the next lower suit.

The other two rules discovered by the program are very poor. They are typical of the kinds of rules that the program discovers when the model does not fit the data very well. Both rules are filled with irrelevant descriptors and values. The current program has very little ability to assess how well a model fits the data. These rules should not be printed by the program, because they are highly implausible.

6 Summary

We have presented here a methodology for discovering sequence-generating rules for the nondeterministic prediction problem. The main ideas behind this methodology are

1. the use of description-space transformations of the initial data and

2. the use of different rule models to guide the search for sequence-generating rules.

Four different description-space transformations (adding attributes, blocking, splitting into phases, and segmenting) and three models (DNF, periodic, and decomposition) have been presented.

The main part of the methodology has been implemented in the program SPARC/E and applied to the NDP problem that arises in the card game Eleusis. The performance of the program indicates that it can discover quite complex and interesting rules.

This methodology is quite general and can be applied to other nondeterministic prediction problems in which the objects in the initial sequence are describable by a set of finite-valued attributes. The main strengths of the method are (a) that it can solve learning problems in which the initial training instances require substantial description-space transformation and (b) that it can search very large spaces of possible rules using a set of rule models for guidance.

Many aspects of this methodology remain to be investigated. We have not considered NDP problems in which (a) the training instances are noisy, (b) the training instances have internal structure so that an attribute vector representation is not adequate, and (c) the sequence-generating rules are permitted to have exceptions. Application of this methodology to real world problems will probably also require the development of additional sequence transformations and rule models. Also, more heuristics need to be developed that can be used to guide the application of transformations and models.

The implementation of the methodology in program SPARC/E and the experiments made so far have demonstrated that the method can discover the majority of Eleusis secret rules playing in ordinary human games. There are some shortcomings of the implementation, however. The program presently conducts a nearly exhaustive depth-first search of the possible models and transformations. Much could be gained by

having the program conduct a best-first heuristically-guided search instead. The present implementation does not include the ability to valuate the plausibility of the rules it discovers. It is also not able to simplify rules by removing redundant selectors, nor is it able to estimate the degree of nondeterminism of the rule. Both of these can be implemented without too much difficulty by including inference routines that make more complete use of the background knowledge already available to the program. Finally, an important weakness of the current program is its inability to form composite models.

In addition to these specific problems, there are some more general problems that further research in the area of sequence-generating laws should address. First, in some real world problems, there are several example sequences available for which the sequence-generating law is believed to be the same. Such problems occur, in particular, in describing the process of disease development in medicine and agriculture. A specific problem of this type that has been partially investigated involves predicting the time course of cutworm infestation in a cornfield and estimating the potential damage to the crop (see [Davis, 1981], [Baim, 1983], and [Boulanger, 1983]). In this problem, several sequences of observations are available—one for each field—and there is a need to develop a sequence-generating law that predicts all of these sequences.

A second general problem for further research is to handle processes in which time is a continuous variable. In particular, programs that could perform qualitative modelling of such processes and qualitative evaluation of trends based on example event sequences would be very valuable. AI research has so far given little attention to these tasks.

Acknowledgments

Earlier versions of this chapter have appeared in the 1983 Machine Learning Workshop and subsequently in the AI Journal. The authors wish the thank the referees of the Workshop and the Journal for their helpful suggestions. Thanks also go to Danny Berlin for bringing several errors to our attention. The authors are grateful to Robert Abbott for inventing the game Eleusis, which was the original motivation for this research and the source of examples for SPARC/E. They also thank Donald Michie for challenging them to develop such a program

A part of the work was done while the second author worked at the Artificial Intelligence Laboratory at Massachusetts Institute of Technology. Support for MIT's Artificial Intelligence research is provided in part by the Advanced Research Projects Agency under Office of Navel Research Contract No. N00014-80-C-0505. The authors gratefully acknowledge the partial support of the National Science Foundation under grant DCR-84-06801 and of the Office of Naval Research under grant No. N00014-82-K-0186.

References

Abbott, R., "The New Eleusis," Available from the author, Box 1175, General Post Office, New York, NY 10116, 1977.

Baim, P. W., "Automated Acquisition of Decision Rules: Problems of Attribute Construction and Selection," M.S. thesis, Department of Computer Science, University of Illinois, Urbana, Illinois, 1983.

Boulanger, A. G., "The Expert System PLANT/CD: A Case Study in Applying the General Purpose Inference System ADVISE to Predicting Black Cutworm Damage in Corn," M.S. thesis, Department of Computer Science, University of Illinois, Urbana, Illinois, 1983.

Buchanan, B. G., and Mitchell, T. M., "Model-Directed Learning of Production Rules," in *Pattern-directed Inference Systems*, Waterman, D. A., and Hayes-Roth, F., (eds.), Academic Press, New York, 1978.

Cohen, P., and Feigenbaum, E. A., *The Handbook of Artificial Intelligence*, Vol. III, Kaufmann, Los Altos, 1982.

Davis, J., "CONVART: A Program for Constructive Induction on Time-Dependent Data," M.S. thesis, Department of Computer Science, University of Illinois, Urbana, Illinois, 1981.

Dietterich, T. G., London, R., Clarkson, K. and Dromey, G., "Learning and Inductive Inference," Chapter XIV in Vol. 3 of *The Handbook of Artificial Intelligence*, Cohen, P. R., and Feigenbaum, E. A., (eds.). 1982.

Dietterich, T. G., and Michalski, R. S., "Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods," *Artificial Intelligence*, 1981.

Engelmore, R., and Terry, A., "Structure and Function of the Crysalis System," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1979.

Friedland, P. E., "Knowledge-Based Experiment Design in Molecular Genetics," Rep. No. HPP-79-29, Department of Computer Science, Stanford University, 1979.

Gardner, M., "On Playing the New Eleusis, the game that simulates the search for truth," *Scientific American*, No. 237, pp. 18-25, October, 1977.

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

Hofstadter, D. R., "The Architecture of JUMBO," *Proceedings of the International Machine Learning Workshop*, Allerton House, University of Illinois, June, pp. 161-170, 1983.

Hofstadter, D. R., "Analogies and Roles in Human and Machine Thinking," In Press.

Karpinski, J., and Michalski, R. S., "A System that Learns to Recognize Hand-written Alphanumeric Characters," *Proce Institute Automatyki, Polish Academy of Sciences*, No. 35, 1966.

Kotovsky, K., and Simon, H. A., "Empirical Tests of a Theory of Human Acquisition of Concepts for Sequential Patterns," *Cognitive Psychology*, No. 4, pp. 39-424, 1973.

Langley, P. W., "Descriptive Discovery Processes: Experiments in Baconian Science," Rep. No. CMU-CS-80-121, Department of Computer Science, Carnegie-Mellon University, 1980.

Lenat, D. B., "The Role of Heuristics in Learning by Discovery: Three Case Studies," in *Machine Learning*, Michalski, R. S., Carbonell, J G., and Mitchell, T. M., (eds.), Tioga, Palo Alto, 1983.

McCarthy, J., "Programs with Common Sense," in *Proceedings of the Symposium on the Mechanization of Thought Processes*, National Physical Laboratory, pp. 77-84, 1958.

McCarthy, J., "Epistemological Problems of Artificial Intelligence," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 1038-1044, 1977.

McCarthy, J., and Hayes, P., "Some Epistemological Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence 4*, Meltzer, B., and Michie, D., (eds.), Edinburgh University Press, Edinburgh, pp. 463-502, 1969.

Michalski, R. S., "On the Quasi-Minimal Solution of the General Covering Problem," in *V International Symposium on Information Processing, FCIP 69*, Yugoslavia, Vol. A3, 2-12, 1969.

Michalski, R. S., "A Variable-valued Logic System as Applied to Picture Description," in *Graphic Languages*, P. Nake and A. Rosenfeld, (eds.), 20-47, 1972.

Michalski, R. S., "Knowledge Acquisition Through Conceptual Clustering: A Theoretical Framework and an Algorithm for Partitioning Data into Conjunctive Concepts," *Journal of Policy Analysis and Information Systems*, Vol. 4, No. 3, pp. 219-244, Sept. 1980.

Michalski, R. S., "A Theory and Methodology of Inductive Learning," *Artificial Intelligence*, Vol. 20, 111-161, 1983.

Michalski, R. S., and Chilausky, R. L., "Learning by Being Told and Learning From Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis," *Policy Analysis and Information Systems*, Vol. 4, No. 2, June 1980.

Michalski, R. S., and Stepp, R. E., "Learning From Observation: Conceptual Clustering," in *Machine Learning*, Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., (eds.), Tioga, Palo Alto, 1983.

Mitchell, T. M., "Version Spaces: An approach to concept learning," Rep. No. STAN-CS-78-711, December, 1978.

Mitchell, T. M., Utgoff, P. E., and Banerji, R. B., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," in *Machine Learning*, Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., (eds.), Tioga, Palo Alto, 1983.

Persson, S., "Some Sequence Extrapolating Programs: A Study of Representation and Modeling in Inquiring Systems," Rep. No. CS50, 1966.

Samuel, A. L., "Some Studies in Machine Learning using the Game of Checkers," in *Computers and Thought*, Feigenbaum, E. A., and Feldman, J., (eds.) McGraw-Hill, New York, pp. 71-105, 1963.

Samuel, A. L., "Some Studies in Machine Learning using the Game of Checkers II—Recent Progress," *IBM Journal of Research and Development*, Vol. 11, No. 6, pp. 601-617, 1967.

Schank, R., and Abelson, R., "Scripts, Plans, and Knowledge," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pp. 151-157, 1975.

Simon, H. A., "Complexity and the Representation of Patterned Sequences of Symbols," *Psych. Review*, Vol. 79, No. 5, 369-382, 1972.

Simon, H. A., and Kotovsky, K., "Human Acquisition of Concepts for Sequential Patterns," *Psychological Review*, Vol. 70, pp. 534-546, 1963.

Solomonoff, R. S., "A Formal Theory of Inductive Inference," *Information and Control*, Vol. 7, pp. 1-22, 224-254, 1964.

Soloway, E. M., "Learning = Interpretation + Generalization: A Case Study in Knowledge-Directed Learning," Rep. No. COINS TR-78-13, Computer and Information Science Dept., U. Mass. at Amherst, 1978.

Winston, P. H., "Learning Structural Descriptions from Examples," Rep. No. AI-TR-231, MIT, 1970.

I. Notational conventions

The following notational conventions are employed in this chapter. In general, lowercase letters denote objects in some sequence (q, ph, b) or index variables (i, j, k) or the lengths of sequences (m, n). Uppercase letters denote sets of objects, attributes, and so on (Q, F, S) as well as parameters of models and transformations (L, P). Small capitals denote attributes (COLOR, RANK) and their values (RED, KING).

$\langle\rangle$	Angle brackets denote sequences of objects, e.g., <2 4 6 8> and also periodic rules, e.g., $\langle[\text{COLOR}(ph_{1,i}) = \text{red}],[\text{COLOR}(ph_{2,i}) = \text{black}]\rangle$.
q_i	The i-th object in an input sequence.
q_i'	The i-th object in a derived sequence.
q_i^-	An object that constitutes an incorrect extension of the sequence after object q_{i-1}.
b_i	The i-th block in a sequence derived by the blocking transformation.
ph_i	The i-th phase derived by the splitting transformation.
$ph_{i,j}$	The j-th object in the i-th phase after a splitting transformation.
n	The number of descriptors.
E	The space of possible events.
F	The starting set of attributes for a transformation.
S	The starting set of sequences for a transformation.
Q	The starting set of objects for a transformation.
F'	The set of derived attributes from a transformation.
S'	The set of derived sequences from a transformation.
Q'	The set of derived objects from a transformation.
F_j	The set of selectors describing object q_{i-j} in block b_i.
g	The sequence-generating function that maps a sequence into a set of objects Q_{k+1} that can appear as continuations of the sequence.
Q_{k+1}	The set of objects that can appear as continuations of the sequence $\langle q_1, q_2, \ldots, q_k\rangle$.
P	The number of phases parameter of the splitting transformation and the periodic model.
L	The lookback parameter of the blocking transformation and all three models.
$[f_i(q_j) = r_k]$	A simple selector, which asserts that feature f_i of object q_j has the value r_k.
$[f_i(q_j) = r_1 \vee r_2 \vee r_3]\}$	A selector containing an internal disjunction. It asserts that f_i can have the value r_1 or r_2 or r_3.
D prefix	The D prefix on an attribute name indicates that it is a difference attribute. Hence, $\text{D-RANK}(q_i,q_{i-1})$ is equal to $\text{RANK}(q_i) - \text{RANK}(q_{i-1})$.

s prefix	The s prefix on an attribute name indicates that it is a summation attribute. Hence, s-RANK(q_i,q_{i-1}) is equal to RANK(q_i) + RANK(q_{i-1}).
$d(F_i, F_j)$	The set of difference selectors obtained by "subtracting" selectors F_j from F_i.
$s(F_i, F_j)$	The set of summation selectors obtained by "adding" selectors F_i and F_j.
$\Rightarrow$	Logical implication.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-85-939	2.	3. Recipient's Accession No.
4. Title and Subtitle Learning to Predict Sequences			5. Report Date February 1985 6.
7. Author(s) T. Dietterich and R. S. Michalski			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF DCR 84-06801 ONR N00014-82-K-0186
12. Sponsoring Organization Name and Address National Science Foundation Washington, DC Office of Naval Research Arlington, VA			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts This paper considers the problem of discovering a rule characterizing a given sequence of events (objects) and able to predict a plausible continuation of the sequence. This prediction is non-deterministic because the rule doesn't necessarily tell exactly which events must appear next in the sequence but rather determines a set of plausible next events. It is assumed that the individual events in the sequence are characterized by a set of attributes and that the next event depends solely on the values of the values of the attributes for the previous events in the sequence. The attributes are either initially given or are derived from the initial ones through a chain of inferences. Three basic rule models are employed to guide the search for a sequence-generating rule: decomposition, periodic, and disjunctive normal form (DNF). The search process involves simultaneously transforming the initial sequences to derived sequences and instantiating general rule models to find the best match between the instantiated model and the derived sequence. A program, called SPARC/E, is described that implements most of the methodology as applied to discovering sequence generating rules in the card game Elcusis. This game, which attempts to model the process of scientific discovery, is used as a source of examples illustrating the performance of SPARC/E.

17. Key Words and Document Analysis.

Machine Learning
Sequence Extrapolation
Inductive Inference
Part-to-Whole Induction
Data Transformation
Model-Directed Learning
Eleusis

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 51
	20. Security Class (This Page) UNCLASSIFIED	22. Price