QUALITATIVE PREDICTION: THE SPARC/G METHODOLOGY FOR INDUCTIVELY DESCRIBING AND PREDICTING DISCRETE PROCESSES

by

R. S. Michalski
H. Ko
K. Chen

Current Issues in Expert Systems, pp. 125-158, London: Academic Press Inc., 1987.

5. Qualitative prediction: The SPARC/G methodology for inductively describing and predicting discrete processes

RYSZARD MICHALSKI, HEEDONG KO
and KAIHU CHEN

Abstract. Qualitative prediction is concerned with problems of building symbolic descriptions of processes, and using these descriptions for predicting a plausible continuation of these processes. It stresses the qualitative form of prediction, as it does not seek precise characterization of future events, but rather a specification of plausible properties and constraints on the future events. An important aspect of qualitative prediction is that only a partial knowledge of the process is available; therefore the construction of a description must necessarily involve inductive inference. It also involves deductive inference to relate the observed process to the concepts contained or derivable from the system's background knowledge.

This chapter describes a domain-independent methodology, SPARC/G, for a simple form of qualitative prediction, where processes are sequences of discrete events or objects that are characterized by finite-valued attributes. Building a description of a process employs general and domain specific knowledge, and involves a new type of inductive learning called *part-to-whole* generalization. The key idea behind the methodology is the use of multiple description models, and model-oriented transformations of the input sequence. Each description model constrains the syntactic form of candidate descriptions, and in this way greatly reduces the total search space. A model is instantiated to a specific description by defining various parameters. A description is considered plausible if it fits a transformed input sequence well, according to the requirements of the model.

The methodology is illustrated by several example problems, such as discovering a secret code for a passage through a sequence of channels, determining preconditions for actions in a blocks world, learning a robot action sequence, predicting the motion of an oscillating spring, and discovering rules in the card game ELEUSIS that models the process of scientific discovery.

1 INTRODUCTION

1.1 What is Qualitative Prediction?

Events in our world tend to be highly interdependent. This interdependence

CURRENT ISSUES IN EXPERT SYSTEMS
ISBN 0-12-714030-1

makes it possible to make predictions about the future on the basis of our knowledge of the past. In fact, the whole purpose of building and maintaining knowledge is to be able to predict and/or influence the future. If our world were a sequence of completely unrelated random scenes, and therefore our knowledge of the past were of no use to interpret or predict future events, there would be little reason for storing any knowledge. As the construction and usage of knowledge is a primary function of intelligence, the need for intelligence would cease also. The above agrees with the observation by Rivest (at a seminar at the Artificial Intelligence Laboratory, MIT, Fall 1985) that "the purpose of intelligence is to predict the future."

The relationship between future and past is usually imprecise and uncertain. Also, it is typically very complex and multifactored. An important way to capture this relationship is to build descriptions or models that are qualitative, i.e. that characterize processes in terms of causal relationships, trends and dependencies. In qualitative prediction the main stress is on building descriptions from partial knowledge of a process. Therefore the major type of inference involved here is inductive. This is different from the approaches in De Kleer and Brown (1984) and Forbus (1984), which are deductive in nature. Inductively derived descriptions may range from statements of "surface" properties (e.g. observable physical properties) to causal explanations and abstract relationships characterizing the process.

The most widely researched type of inductive learning has been concerned with discovering a general description of a class of objects, given selected instances of the class. For example, given instances of cancerous and noncancerous cells, the task is to determine a general rule for discriminating between these two types of cells (Michalski, 1983). This type of inductive learning is called **instance-to-class** generalization.

The inductive learning involved in qualitative prediction is different from such instance-to-class generalization. It involves a form of the **part-to-whole** generalization. To explain the latter type of induction, let us consider a few examples. Suppose that a palaeontologist has excavated bones of a prehistoric animal, and from his information he then hypothesizes the entire skeleton of the animal. As another case, consider an archaeologist who is given an incomplete set of pieces of a broken ancient sculpture, and has to reconstruct the original. In such cases we do not have independent examples of some class of objects, but rather interdependent parts of one structured object. The task is to hypothesize a description of the whole object.

Clearly, the above problems fit the general notion of inductive generalization, but are not the **instance-to-class** generalization problems. In **instance-to-class** we are given instances that are **independent** members of a class; any possible relations among training instances are considered irrelevant. In **part-to-whole** generalization, the inputs are descriptions of **parts** of a structured object, and relations among the parts are of primary importance.

A very simple form of the **part-to-whole** generalization problem occurs in IQ tests where the task is to predict a plausible continuation of a sequence of numbers or letters. The given sequence can be viewed as a part of an unknown complete sequence. The task is to hypothesize the remaining part of the complete sequence on the basis of the known parts of the sequence.

Suppose that instead of letters or numbers, we have snapshots of some process occurring in time. Assume also that our background knowledge contains sufficient information for characterizing the relationships between these snapshots. The task is to determine a description of the process that not only accounts for snapshots seen so far but also suggests a plausible continuation of this process. Suppose further that the description sought is not quantitative but rather qualitative. Instead of precise prediction of the future process, which may not be possible, one desires only a general characterization of the properties that the future events are expected to satisfy. In this exploratory paper we assume that a process is represented by a sequence of events, called an **episode**:

$$E = \langle e_1, e_2, e_3, \ldots, e_k \rangle.$$

It is also assumed that each event can be satisfactorily characterized by a vector of values of certain attributes:

$$x_1(e_j), \quad x_2(e_j), \quad x_3(e_j), \quad \ldots, \quad x_n(e_j),$$

or briefly,

$$x_1, \quad x_2, \quad x_3, \quad \ldots, \quad x_n,$$

We shall also assume that attributes $x_1, \ldots, x_n$ have domains that are known *a priori* (value sets):

$$D(x_1), \quad D(x_2), \quad D(x_3), \quad \ldots, \quad D(x_n).$$

Each $D(x_i)$ is the set of all values an attribute can possibly take for any event in the given or future episodes. These value sets, their structure (which defines the type of an attribute), the constraints on the relationships among attributes, and knowledge of the application domain, constitute the *background knowledge* of a qualitative prediction system.

Given an episode E and the background knowledge, the task is to induce a description that characterizes the given episode, and predicts plausible future events; i.e. $e_{k+1}, e_{k+2}, \ldots$ Such a description is called a *qualitative prediction rule* (QPR). It is not required that a QPR specify precisely what event will follow, but merely that it constrain the type of events that may follow. When constraints are sufficiently strong that only one event may satisfy them at each place then the QPR is a deterministic prediction rule; otherwise, it is a nondeterministic prediction rule. Discovering such qualitative prediction rules is called a nondeterministic prediction problem (NDP).

An example of an NDP problem is to discover the secret rule in the card game ELEUSIS. The rule, known only to the dealer, describes a sequence of cards that are *legal*. Players attempt to play one or more cards that correctly extend the sequence. To do so, they have to infer the secret rule or its approximation from the cards observed so far. Dietterich (1980) describes a method and a program for discovering such rules which in some instances outperformed human players. Another paper (Michalski *et al.*, 1985); describes the SPARC/E program that discovers rules, and plays the Eleusis game as an autonomous player using the rules discovered. The methodology underlying the SPARC/E program was subsequently generalized and described by Dietterich and Michalski (1985). This paper further expands and extends the method, and presents results of various experiments with an implemented program, SPARC/G (which stands for "Sequential Pattern Recognition/General). These results demonstrate the performance and generality of the method.

Three main topics are discussed in this chapter. First, various models for expressing descriptions are defined, and algorithms for constructing descriptions based on these methods are detailed. Secondly, a program that implements the methodology is described. Finally, several example problems are used to demonstrate the strengths and weaknesses of the methodology.

1.2 Relationship to Time-Series Analysis

There are parallels between this approach and the regression and spectral methods in time-series analysis (Box and Jenkins, 1976). Regression methods attempt to explain the behaviour of a particular variable in terms of the behaviour of a set of independent variables using a polynomial regression function. Spectral analysis attempts to describe the behaviour of a particular variable by analysing its frequency spectrum. In our approach, we use three description models. Our decomposition model corresponds to the regression polynomial. Our periodic model is a symbolic counterpart of the spectral method. However, our third model, the disjunctive model, seems to have no counterpart in classical time-series analysis. The major differences between the proposed approach and time-series approach can be characterized as follows.

(i) In the proposed methodology, each event in the process can be characterized by a large number of attributes. The attributes may have different types: numerical, nominal, cyclic or structured (where the value set is a hierarchy).

(ii) The prediction for the next events is qualitative and nondeterministic; the system constructs a symbolic description that characterizes the set of plausible next events.

(iii) The background knowledge of the program contains constructive induction rules that generate new attributes not present in the initial data.

We assume that the input information about a given process, and the information derivable from the program's background knowledge, are sufficient for predicting a plausible continuation of the process.

2 INDUCING GENERAL DESCRIPTIONS FROM EPISODES

This section presents the theoretical background and basic algorithms underlying the SPARC/G methodology.

2.1 Events and Episodes

The goal of the SPARC/G methodology is to construct a description of an observed process that permits one to predict qualitatively plausible future events. The desired description should be conceptually simple, and consistent with the information known about the process and the system's background knowledge. To develop such a description, "snapshots" of the process are taken. In each snapshot, we measure the state of the process in terms of various attributes believed to be relevant ("attribute" and "variable" are used interchangeably throughout).

A collection of measurements of the process in one snapshot is called an **event**. A sequence of events in chronological order is called an **episode**.

2.2 Representation of Events

A simple representation of an event is just a list of values of some attributes. A more elaborate representation would be in the form of graphs or predicate logic expressions. Here, we use a representation based on VL_1 (the Variable-Valued Logic 1: Michalski, 1974). Each event is represented by a conjunction of relational statements called **selectors**. Each selector describes some measurements taken from the original process. Conjunctions of selectors are called **VL_1 complexes**, or simply **complexes**. Formally, a **selector** consists of an attribute name, a set of values called a **reference**, and a relation between the attribute name and the set of values. It is written as

[attribute relation reference]

For example, the relation

[suit = clubs v diamonds]

states that the attribute **suit** may take on the value **clubs** or **diamonds**.

Each attribute is assigned an explicit set of values called its **domain**. All legal values in the reference of a selector must be taken from the domain. Four types of attributes are distinguished: linear, nominal, cyclic and structured. Both linear and cyclic attributes have integer values. Nominal attributes have nonordinal values. For example, the domain of the nominal attribute **suit** is {clubs, diamonds, hearts, spades}. A **complex** (a conjunction of selectors) is written by placing selectors adjacent to each other. For example, the complex [suit = clubs v diamonds][value < 3] describes the set of cards {AC, 2C, AD, 2D}. A structured attribute represents a value hierarchy that is built on top of existing attributes, and can be either linear or nominal. For example, the structured attribute **color** (of cards) can be defined using attribute **suit**, such that [color = red] is defined as [suit = hearts v diamonds], and [color = black] is defined as [suit = clubs v spades].

2.3 Representation of Episodes

Subscripts are used to indicate the relative ordering between events. Attributes with subscript 0 refer to the current event of interest. A subscript 1 refers to the event immediately preceding the current event of interest; a subscript 2, to the event before that, and so on. For example, the complex [color1 = red][value0 > 6] states that the color in the preceding event was red and the value in the current event is greater than 6. We also introduce difference and sum attributes. The attribute dvalue01 is defined as value0 − value1. The attribute svalue01 takes on value0 + value1.

2.4 Lookback and Periodic Descriptions

Statistical prediction methods specify possible next values of some attributes along with a probability of each value. The method described here differs from such methods in that it specifies a symbolic description characterizing all possible next events. There are two basic types of descriptions used to characterize a sequence and predict its future course: **lookback** descriptions and **periodic** descriptions. A lookback description is a function *F* of the *lb* most recent events, where *lb* is the lookback parameter. This function predicts the next event, or a set of plausible next events (the nondeterministic prediction) in terms of the properties of the *lb* past events. Thus, given an episode

$$E = \langle e_1, e_2, e_3, \ldots, e_n \rangle,$$

we have

$$F(e_{i-lb}, e_{i-(lb-1)}, \ldots, e_{i-2}, e_{i-1}) = \{e_i\},$$

where $\{e_i\}$ is the set of plausible next events.

An example of a lookback description with $lb = 4$ is the function

$$x_i = x_{i-1} \cdot x_{i-2} - x_{i-3} \cdot x_{i-4}, \qquad \text{where } x_0, x_1 = 1, x_2 = 2, x_3 = 3,$$

that describes the sequence

$$\langle 0, 1, 2, 3, 6, 16, 90, \ldots \rangle.$$

A *periodic description* characterizes a sequence by observing a regularity that binds the events at some fixed distance from each other (the period length) throughout the whole sequence. The relative position of an event within the same period is called a **phase**. For example, the sequence

$$\langle \text{a, b, c, b, c, d, c, d, e, d, e, f}, \ldots \rangle$$

is characterized by a periodic description of period length 3, in which letters of the same phase grow alphabetically.

2.5 Description Models

Inductive learning is the process of generating hypotheses that are plausible in explaining the observed events and useful in predicting the unobserved. One approach to induction is to identify one or more description models that constrain the form of hypothesized descriptions. Inductive learning then becomes a two-step process of first instantiating the model to generate a specific description, and then evaluating the plausibility and utility of the resulting description. Simple forms of such techniques have long been used in traditional regression analysis, where a typical model is a regression polynomial, and statistical tests are used to test the fit between the data and the instantiated model.

Examples of symbolic description models are the decision tree used by Hunt (1966), and the disjunctive normal form used by Michalski (1969, 1971, 1974). Such models carry a good deal of implicit problem-specific knowledge. It is important that a general inductive tool permit dynamic specification, modification and manipulation of the models.

Our method uses three description models.

(1) *Periodic conjunctive model.* This model specifies that the description must be a periodic description in which each phase is described by a single complex. For example, the rule

Period ([color0 = red], [color0 = black])

describes an alternating sequence of red and black cards. Furthermore, we can imagine a periodicity within the phase, in which case we have an embedded periodic rule. For example, suppose that the first phase of the above rule is another periodic sequence of face and nonface cards. This is represented as

Period ([color0 = red][Period ([face0 = true], [face0 = false])], [color0 = black])

(2) *Lookback decomposition model.* This model specifies that the description must be a lookback description in the form of a set of if–then rules:

[color1 = red] $\rightarrow$ [value0 <5]
[color1 = black] $\rightarrow$ [value0 $\geqslant 5$]

The left-hand sides, or condition parts of the rules refer to no more than *lb* (the lookback parameter) events prior to the event to be predicted (subscripts 1, 2, etc.). The right-hand sides provide predictions for the next events in the sequence given that the condition part is true. The decomposition model requires that the left-hand sides be disjoint so that only one if–then rule be applicable at one time.

(3) *Disjunctive normal form (DNF).* This model requires only that the description of the sequence must be a disjunction of VL_1 complexes. For example, the DNF expression

[dsuit01 = 0] v [dvalue01 = 0]

states that either the suit of the current card must be the same as the suit of the previous card, or the value of the current card must be the same as the value of the previous card.

From a logical standpoint, any decomposition rule or periodic rules can be written in disjunctive normal form. The periodic and decomposition models are useful not because of their theoretical expressiveness or power, but because of their assistance in locating plausible descriptions quickly. Depending on the number of descriptive attributes used, the space of all DNF descriptions could be immense and thus difficult to search. Therefore, this is a "catch-all" model, used after the other models have failed.

2.6 Descriptions Based on Segmentation

Often sequences of events are best described in a hierarchical fashion as series of subsequences. For example,

$$S = \langle 3, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 7 \rangle$$

is best described as a sequence of subsequences. Each subsequence is a string of identical digits. The length of each subsequence is one longer than its predecessor. The digit used in the subsequence is one larger than the digit used in the previous subsequence. In our method, this is indicated by a two-part description in which one part defines the segmentation condition, and the second part defines the relations among segments:

Segmentation condition:
String: [dvalue01 = 0]
Intersegment relation:
[dvalue01 = +1][dlength01 = +1]

The segmentation condition defines subsequences of events with constant value (dvalue01 = 0). The intersegment relation defines relations among the segments in the new sequence. For example, dvalue01 and dlength01 refer to the values and lengths of the segments. In our example the sequence is **segmented** into strings of maximal length satisfying this segmentation condition. This yields a new sequence

$$S' = \langle (3,1), (4,2), (5,3), (6,4), (7,5) \rangle.$$

In the original episode S each event of the episode is an entity with only one attribute, the value. In S' each event is related to a subsequence of events in S. Some of the attributes of S may also be used in S', while some others are newly created for S'. For example, the second event in S' has value 4 because all the corresponding events in S have value 4. Events in S' have a new attribute, length, indicating the number of events corresponding to this event in S.

Any description model listed in Section 3.3 can be applied to a sequence after it has been segmented. The discovery of such segmented descriptions requires both the discovery of the segmentation condition and the formulation of the description of the segmented sequence. In the current implementation, the system is equipped with a repertoire of segmentation conditions. A segmentation condition is chosen if its application produces a sufficient (according to a user defined criterion) number of elements in the transformed sequence.

3 THE ALGORITHMS UNDERLYING THE SPARC/G PROGRAM

3.1 Input Representation

The input episode is represented as a list of events. Each event in the list is represented by a set of attributes which are defined by the user. In addition, each event is marked as a positive or negative event of the episode. Let us use a very simple example (Figure 1) to illustrate the workings of SPARC/G. Each event in the episode is characterized by its texture, orientation (in degrees) and size. The representation (what is actually used by the program) is shown in Table 1.

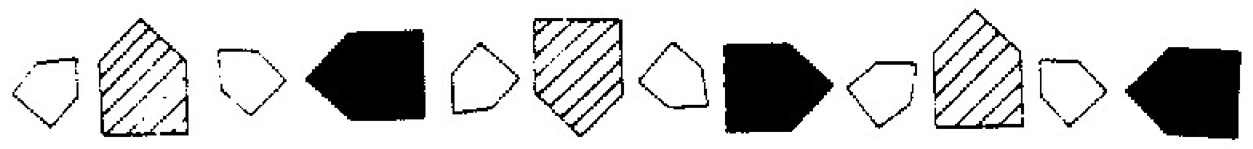
Figure 1 Sequence of geometric shapes.

Table 1 Input VL_1 events.

Event	txtr0	orient0	size0
1	blank	45	small
2	striped	90	big
3	blank	135	small
4	solid	180	big
5	blank	225	small
6	striped	270	big
7	blank	315	small
8	solid	0	big
9	blank	45	small
10	striped	90	big
11	blank	135	small
12	solid	180	big

3.2 Data Transformations

The first step is to use constructive induction rules to derive additional attributes that may be useful for creating descriptions of the episode. Such rules are a part of the program's background knowledge, supplied by the user. New attributes are defined in terms of existing attributes, which in turn may be derived from previously defined attributes. The new attributes

Table 2 Augmented VL_1 events.

Event	txtr0	orient0	size0	shaded0
1	blank	45	small	false
2	striped	90	big	true
3	blank	135	small	false
4	solid	180	big	true
5	blank	225	small	false
6	striped	270	big	true
7	blank	315	small	false
8	solid	0	big	true
9	blank	45	small	false
10	striped	90	big	true
11	blank	135	small	false
12	solid	180	big	true

augment the current event descriptions. Here, a new attribute **shaded** is added that has two values: true and false. The value **false** characterizes a blank texture and the value **true** characterizes any other texture. If the generated attributes pass a preliminary relevance test, they are used to augment episode representation. Such an augmented representation is shown in Table 2.

The second step involves segmenting the episode. As discussed in Section 2, a segmentation condition is a relation that must hold between adjacent events of the segment. SPARC/G segments the episode into strings of maximal length that satisfy the segmentation condition, and then evaluates the potential usefulness of the segmentation. For example, the segmentation is not considered potentially useful if the segmented episode has nearly the same number of events as the original episode, or if the whole episode satisfies the segmentation condition.

The next transformation step involves making the order of the events explicit in the events. If the lookback parameter is one or more, the episode is transformed by augmenting each event with previous events falling within the lookback parameter window. Table 3 is the result of such a transformation derived with null segmentation condition and a lookback of one, then augmented with difference attributes. Now, the episode goes through model specific transformations explained in the next section.

3.3 Model-Dependent Rule Generation

This section explains how each description model is used in searching for a qualitative prediction rule.

Table 3 Transformed VL_1 events.

Event	txtr1	orient1	size1	shaded1	txtr0	orient0	size0	shaded0	dtxtr01	dorient01	dsize01	dshaded01
1	blank	45	small	false	striped	90	big	true	1	45	1	1
2	striped	90	big	true	blank	135	small	false	1	45	1	1
3	blank	135	small	false	solid	180	big	true	1	45	1	1
4	solid	180	big	true	blank	225	small	false	1	45	1	1
5	blank	225	small	false	striped	270	big	true	1	45	1	1
6	striped	270	big	true	blank	315	small	false	1	45	1	1
7	blank	315	small	false	solid	0	big	true	1	45	1	1
8	solid	0	big	true	blank	45	small	false	1	45	1	1
9	blank	45	small	false	striped	90	big	true	1	45	1	1
10	striped	90	big	true	blank	135	small	false	1	45	1	1
11	blank	135	small	false	solid	180	big	true	1	45	1	1

3.3.1 *Rule generation of the decomposition model*

The decomposition model describes an episode by a sequence of production rules. It accepts as input a set of positive events with, optionally, a set of negative events. Some attributes are designated as "left-hand-side" attributes. A decomposition seeks to explain current events in terms of the values of "left-hand-side" attributes. A decomposition-model-based description for the events in Table 3 would be

[shaded1 = true] → [txtr0 = blank][shaded0 = false]
[shaded1 = false] → [txtr0 = solid v striped][shaded0 = true]

This description **decomposes** events on attribute shaded1. It breaks the description of the episode into two if–then rules. The → can be interpreted as an implication. The decomposition algorithm assumes that both the left-hand and right-hand parts of the if–then rules must be single VL_1 complexes, and that the left-hand sides must be logically disjoint.

The decomposition algorithm starts by performing a trial decomposition on each possible left-hand-side attribute. A trial decomposition for a left-hand-side attribute is formed by creating a complex for each value of the attribute occurring in the episode. The complex is formed by merging (set union) the references of corresponding selectors of all events following the left-hand-side attribute. For example, using the events of Table 3, trial decompositions could be performed on txtr1, orient1, size1 and shaded1, but for simplicity Figure 2 represents a decomposition in terms of txtr1 and shaded1. The general idea is to form trial decompositions, choose the best decomposition, and break the problem into subproblems, one for each if–then rule in the selected decomposition. The algorithm can then be applied recursively until a consistent description has been developed.

Figure 2 shows the raw trial decompositions. These are very-low-generality descriptions. They must be processed further before a decision can be made as to which decomposition is best and should be further investigated. Three processing steps are applied to the trial decompositions.

The first processing step involves linear and cyclic interval attributes. These attributes often have many values, and raw trial decompositions based on them may be uninteresting and implausible. An attempt is made to apply the "close interval" inductive inference rule on the left-hand side of the trial decomposition (Michalski, 1983). The algorithm operates by computing distances between adjacent if–then rules, and looking for sudden jumps in the distance measure. Where a jump occurs (a local maximum), the algorithm tries to split the domain into cases.

The distance computation is a weighted multiple-valued Hamming distance. The weights are determined by taking user-specified plausibilities for

Decomposition on txtr1:

[txtr1=solid]	→	[txtrO=blank][orientO=45 v 225][sizeO=small][shadedO=false] [dtxtrO1=1][dorientO1=45][dsizeO1=1][dshadedO1=1]
[txtr1=blank]	→	[txtrO=solid v striped][orientO=0 v 90 v 180 v 270][sizeO=big][shadedO=true] [dtxtrO1=1][dorientO1=45][dsizeO1=1][dshadedO1=1]
[txtr1=striped]	→	[txtrO=blank][orient=45 v 135 v 315][sizeO=small][shadedO=false] [dtxtrO1=1][dorientO1=45][dsize=1][dshadedO1=1]

Decomposition on shaded1:

[shaded1=true]	→	[txtrO=blank][orientO=45 v 135 v 225 v 315][sizeO=small][shadedO=false] [dtxtrO1=1][dorient=45][dsizeO1=1][dshadedO1=1]
[shaded1=false]	→	[txtrO=striped v solid][orientO=0 v 90 v 180 v 270][sizeO=big][shadedO=true] [dtxtrO1=1][dorientO1=45][dsizeO1=1][dshadedO1=1]

Figure 2 Trial decompositions.

Decomposition on txtr1:

[txtr1=solid]	→	[txtrO=blank][orientO=45..225][sizeO=small][shadedO=false] [dtxtrO1=1][dorientO1=45][dsizeO1=1][dshadedO1=1]
[txtr1=blank]	→	[txtrO=solid v striped][orientO=0..270][sizeO=big][shadedO=true] [dtxtrO1=1][dorientO1=45][dsizeO1=1][dshadedO1=1]
[txtr1=striped]	→	[txtrO=blank][orient=45..315][sizeO=small][shadedO=false] [dtxtrO1=1][dorientO1=45][dsize=1][dshadedO1=1]

Decomposition on shaded1:

[shaded1=true]	→	[txtrO=blank][orientO=45..315][sizeO=small][shadedO=false] [dtxtrO1=1][dorient=45][dsizeO1=1][dshadedO1=1]
[shaded1=false]	→	[txtrO=striped v solid][orientO=0..270][sizeO=big][shadedO=true] [dtxtrO1=1][dorientO1=45][dsizeO1=1][dshadedO1=1]

Figure 3 Generalized trial decompositions.

each attribute and relaxing these weights according to the discriminating power of each attribute (taken singly). For instance, if right-hand-side attribute is irrelevant in some if–then rules, i.e., its reference contains all possible values, then its weight is reduced to zero. The distances between adjacent if–then rules are computed and local maxima are located. If there is one maximum then the interval is split there, and two if–then rules are created. If there are two maxima then there are three intervals, and each creates one if–then rule. If there are more than two maxima then the smaller maxima are suppressed. Similar techniques are used for cyclic interval domains.

Once the cases have been determined, each trial decomposition is processed by applying the domain type specific rules of generalization to the selectors on the right-hand sides of the if–then rules. The "close interval" inference rule is applied to linear and cyclic attributes. Special domain types are defined for difference attributes (attributes derived by subtracting two other attributes). The rules of generalization for difference attributes attempt to find intervals about the zero point of the domain. Thus [dvalue01 = −3 v 1 v 2] would be generalized to [dvalue01 = −3.. + 3]. One-sided intervals away from zero are also created: [dvalue01 = 3 v 4 v 6] would be generalized to [dvalue01 > 0]. These generalizations are only performed if the reference contains more than one value. Corresponding to the trial decompositions of Figure 2 we get the generalized trial decompositions shown in Figure 3. The notation [size0 = *] is used when an attribute can take on any value from its domain.

The third processing step examines the different if–then rules and attempts to make the right-hand sides of the rules disjoint by removing selectors whose references are overlapping among them. Figure 4 shows the results of this step.

Decomposition on txtr1:

[txtr1 = solid]	$\rightarrow$	Any Event
[txtr1 = blank]	$\rightarrow$	Any Event
[txtr1 = striped]	$\rightarrow$	Any Event

Decomposition on shaded1:

[shaded1 = true]	$\rightarrow$	[txtr0 = blank][shaded0 = false]
[shaded1 = false]	$\rightarrow$	[txtr0 = solid v striped][shaded0 = true]

Figure 4 Trial decompositions with overlapping selectors removed.

The selection of the best decomposition uses a set of cost functions that measure characteristics of each trial decomposition. The cost functions are as follows.

(1) Count the number of negative examples that are incorrectly covered by this decomposition.

(2) Count the number of cases (if–then rules) in this decomposition.

(3) Return the user-specified plausibility for the attribute being decomposed on.

(4) Count the number of null cases for this decomposition

(5) Count the number of "simple" selectors in this decomposition. A simple selector can be written with a single value or interval in the reference (e.g. [value01 > 4] is a simple selector). After applying the generalization rules (as in Figure 3) all selectors except those with nominal attributes are simple.

The cost functions are applied in an ordered fashion using the lexicographic sort algorithm developed by Michalski (1980). The trial decomposition with the lowest cost is selected. The lowest cost solution is the decomposition on shaded1 shown in Figure 4. It states that if the figure is shaded then the texture of the next figure is blank and its shade is not shaded. And if the figure is not shaded then the texture of the next figure is solid or striped and is shaded.

Once the best trial decomposition has been selected, it is checked to see if it is consistent with the events (covers no negative events). If so, the decomposition algorithm terminates. If it is not then the problem is decomposed into separate subproblems, one for each if–then rule in the selected decomposition. Then the algorithm is repeated to solve these subproblems. (The subproblems are solved simultaneously, not independently.)

The strengths of the decomposition algorithm are as follows.

(1) Speed—good decompositions are located quickly.

(2) Transparency—decomposition descriptions are easy to interpret.

(3) Generality—the algorithm can discover a large class of symbolic relations between the current event and past events within a given lookback.

3.3.2 *Rule generation using the periodic model*

The periodic model is used to test if events in the episode display a periodic behaviour. It is assumed that the parameter defining the number of phases is provided to the algorithm. In searching for a periodic description, the system may try different values of this parameter. Each phase is treated in a manner similar to the treatment of the different if–then cases in the trial decomposition algorithm described earlier. First, the events in each phase are combined to form a single complex (by forming the union of references of corresponding selectors). For the episode in Figure 1, using a phase of two, the results are

Phase1: [txtr0 = blank][orient0 = 45 v 135 v 225 v 315]
[size0 = small][shaded0 = false]
Phase2: [txtr0 = solid v striped][orient0 = 0 v 90 v 180 v 270]
[size0 = big][shaded0 = true]

Note that in order to simplify descriptions, no difference attributes or attributes describing previous events are included in these derived events. First, overlapping complexes are dropped. Complexes that do not cover examples of other phases or negative examples are then generalized further:

Phase1: [txtr0 = blank][orient0 = 45..315]
[size0 = small][shaded0 = false]
Phase2: [txtr0 = solid v striped][orient0 = 0..270]
[size0 = big][shaded0 = true]

If these generalized complexes still do not cover negative examples, selectors with overlapping references (overlapping with selectors in other phases) are removed:

Phase1: [txtr0 = blank][size0 = small][shaded0 = false].
Phase2: [txtr0 = solid v striped][size0 = big][shaded0 = true]

If these complexes are still consistent, they are returned as the final description.

Both the periodic and the decomposition algorithms go through the above postprocessing steps until the description becomes inconsistent, at which time the algorithm backs up and returns the version of the description before it was overgeneralized to become inconsistent. In some cases, the star generation process of the Aq algorithm is invoked to attempt to extend the description against negative examples and examples of other phases.

For each phase from the above, a new episode is assembled. This episode is considered a full-fledged episode so that the periodic algorithm is invoked recursively until either the newly assembled episode is trivial, such as having length of one, or the description returned from the next call to the model is implausible. For the example, the episode for the second phase is again periodic:

Phase21: [txtr0 = striped]
Phase22: [txtr0 = solid]

Here the second phase of the top level is described by an embedded periodic rule of two phases, Phase21 and Phase22. The full recursive periodic description is

```
Period([txtr0 = blank][size0 = small][shaded0 = false],
       [size0 = big][shaded0 = true]
       [Period([txtr0 = solid], [txtr0 = striped])])
```

This rule states that the episode has two phases: the events in the first phase have "blank" texture, "small" size and shaded; the events in the second phase have "solid" or "striped" texture, "big" size and not-shaded, also the textures alternate from striped to solid.

3.3.3 Rule generation using the DNF model

The DNF (disjunctive normal form) model employs the Aq algorithm (Michalski, 1969, 1971), which was originally developed in the context of switching theory and subsequently used for inductive inference (Michalski, 1972, 1973). The algorithm accepts as input a set of positive events and a set of negative events, and produces an optimized cover of the positive events against the negative events. Such a cover is a description that is satisfied by all of the positive events, but by none of the negative events. The process of developing a cover involves partially computing the complement of the set of negative events and intelligently selecting complexes which cover positive events. The final cover may be a single complex or a disjunction of complexes. Aq seeks to develop covers that satisfy predefined criteria, such as minimizing the number of complexes in the cover, the total cost of attributes, etc.

The algorithm proceeds in best-first fashion by the method of disjoint stars. A positive event **e1** is determined, and a star is built about **e1**. A **star** is the set of all maximally general complexes that cover **e1** and do not cover any negative event. The best complex in the star, **lq**, is chosen and included in the goal description. All events covered by **lq** are removed from further consideration. The above process is then repeated. However, the newly selected **e1** must not be covered by any element of *any previous star*. In this manner the algorithm builds disjoint, well-separated stars. It has been shown that the number of such stars is a lower bound on the minimum number of complexes in any cover (Michalski, 1969). The process repeats until all events are covered by at least one **lq** complex. Disjunctions of the selected complexes forms the goal discription. Some clean-up operations are required in the case where some positive events were covered by some star, but by no **lq**.

A simplified description of the process of building a star about an event **e1** is given as follows: each negative event is complemented, and then multiplied out, with the proviso that each resulting complex must cover **e1**. After each event is multiplied out, the set of intermediate products (so-called partial stars) is trimmed according to a user-specified preference criterion, and only the *MAXSTAR* best elements are retained. The final star has at most *MAXSTAR* elements in it.

Note that all of the steps mentioned (complementation, multiplication, etc.) are performed on attributes which can take on a set of values. This is a multiple-valued covering process.

The strengths of the algorithm include the following.

(1) Quasi-optimality—the algorithm efficiently generates covers that are optimal or near-optimal.

(2) Flexibility of cover optimality and type—the user can specify the cover optimality criterion that reflects the specific aspects of the problem. The criterion determines which **lq** is chosen from each star and which partial stars are retained during the star-building process. The algorithm can also be told the type of the cover sought. The cover can be disjoint (complexes are disjoint), intersecting (complexes are overlapping) or ordered (complexes are linearly ordered).

(3) Optimality estimate—if no trimming is performed, the algorithm provides an estimate of the maximum difference between the number of complexes in the solution and in the minimum solution.

The DNF model is used to discover properties that describe the collection of all positive/negative events. Sequential information, if any, exists in the form of attributes that characterize the relationship between events. The Aq algorithm is given the set of all positive events and negative events augmented with the derived attributes. The algorithm then attempts to find descriptions that describe all positive events, but none of the negative events. With orientation defined as a cyclic attribute so that zero degree is considered to be 45 degrees "larger" than 315 degrees, a difference attribute **dorient01** can be defined. Given appropriated negative events (not shown in Figure 1), a description

$$[\text{dorient01} = 45]$$

is discovered as the description that perfectly characterizes the positive events.

3.4 Description Evaluation and Selection

This phase examines rules developed by the above induction algorithms in order to filter out redundant information in the generated rules. For example, the following are the rules given in Figure 4:

[shaded1 = dark] → [txtr0 = blank][shaded0 = false]
[shaded1 = clear] → [txtr0 = solid v striped][shaded0 = true]

Note that [txtr0 = blank] is equivalent to [shaded0 = false], and [txtr0 = solid v striped] is to [shaded0 = true]. This redundancy was caused because the induction algorithms were not aware of the structural relationships between attributes. This redundancy is removed by the following procedure:

```
for each rule in the rulebase do
  for each complex in the rule do
    for selectors $A$ and $B$ in the complex, and both $A$ and $B$ are based
    on some attribute do
      if they are equivalent then keep the syntactically simpler one
      else if $A \subset B$ then drop $A$
           else if $B \subset A$ then drop $B$
```

If A and B are based on two different attributes then A and B cannot be redundant. For example, shaded0 and dshaded01 cannot be redundant, since shaded0 is based on txtr0 while dshaded01 is based on both shaded0 and shaded1.

When an episode is segmented, some additional operations may be required. For example, given the episode

$$S = \langle 3, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7 \rangle,$$

one would not want to create a segment for the sevens. Such a segment would indicate that there is a string of sevens of length 2. If the induction algorithms received such an event, they would not be able to discover that the length of a string always increases by 1. So the segmentation process must leave the end of the episode unsegmented. Each description produced by the induction algorithm must be checked to verify that it is consistent with the tail end of the episode.

Finally the plausibility of the descriptions is assessed. First of all, the rule must be consistent; that is, it should not predict incorrectly events within the episode. Another criterion for plausibility is that the rule should be conceptually simple. This is approximated in the program by measuring syntactic complexity of the rule, such as the number of values in a reference, the number of selectors in each complex, the number of complexes in the rule and so on.

4 APPLICATIONS

This section presents results from applying the SPARC/G program (written in Berkeley PASCAL, running under Unix 4.2 BSD on a Sun-2/120 workstation) to a few example problems. Possible improvements and extensions to the program are also suggested.

Example 1: Discover Safe Passage through Channels

Suppose that two oceans Ocean$_1$ and Ocean$_2$ are connected by a network of channels, and the passageways are full of mines. The mines are regularly activated or deactivated by the enemy through remote control. The enemy signals the safe passageway to its ships by left and right beacons located before and after the junctions. The colour and frequency of the beacon are governed by a secret code indicating the safe passage. The ally observes the enemy ships passing from Ocean$_1$ to Ocean$_2$, and would like to discover the code so that its ships can also pass through the channels safely. SPARC/G was given the following descriptors:

(1) LeftColor (colour of the left beacon): {green, red, blue};

(2) RightColor (colour of the right beacon): {green, red, blue};

(3) LeftFrequency (frequency of the left beacon): {low, medium, high};

(4) RightFrequency (frequency of the right beacon): {low, medium, high}.

A map of the channel is given in Figure 5. The routes not taken by the enemy are considered unsafe, and are marked as an arrow with a bar across it. To discover the rule that characterizes the safe passage, it is hypothesized that all relevant information for the secret code is provided by the attributes of the beacons before and after each junction. The input episode is given in Table 4.

Table 4 Input events for Example 1.

Event	LeftColor	RightColor	LeftFrequency	RightFrequency	Route
1	red	green	medium	medium	taken
2	green	blue	high	low	not-taken
3	green	blue	low	high	taken
4	red	green	high	high	not-taken
5	blue	red	medium	medium	taken
6	red	blue	medium	high	not-taken
7	red	green	medium	low	not-taken
8	green	red	low	medium	taken
9	blue	red	medium	high	not-taken
10	red	red	high	low	taken
11	green	blue	medium	low	not-taken
12	green	blue	high	low	not-taken
13	blue	green	low	medium	taken
14	blue	green	low	high	not-taken
15	blue	red	low	medium	taken

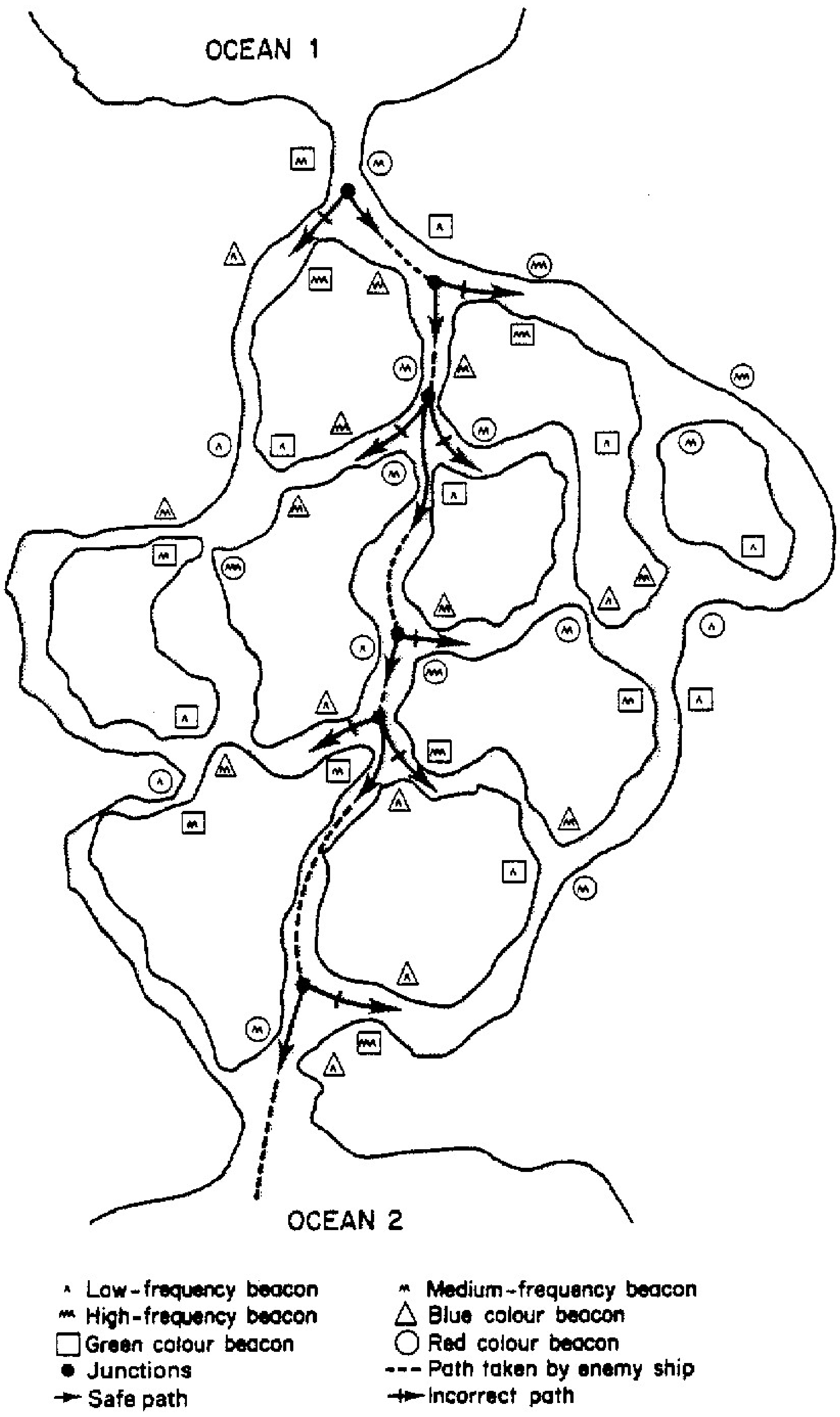

Figure 5 Mined channels.

The program discovered the following safe-passage rule using the decomposition rule model:

Rule 1: decomposition model, lookback: 1, nphases: 0
[LeftColor-before = red] → [RightFrequency-after > RightFrequency-before]
[LeftColor-before = green] → [RightFrequency-after < RightFrequency-before]
[LeftColor-before = blue] → [RightFrequency-after = RightFrequency-before]

The computation time was approximately one second on a Sun-2/120. The rules can be paraphrased as follows. The passage is safe if

- the colour of the left beacon before a junction is red, and the frequency of the next beacon on the right is lower than that of previous beacon on the right; or
- the colour of the left beacon before a junction is green, and the frequency of the next beacon on the right is higher than that of previous beacon on the right; or
- the colour of the left beacon before the junction is blue, and the frequency of the next beacon on the right is the same as that of previous beacon on the right.

In the paraphrase, the "implication" is interpreted as "and". This is allowed because the left-hand sides of the implication in decomposition rules are disjoint and complete with respect to the domain of the attribute. The program discovered the rules that are exactly the ones used to generate the example of safe passage. This is a very satisfactory result.

Example 2: Learning Preconditions in a Blocks World

In many planning systems, operations are often expressed as precondition–postcondition pairs. Preconditions specify the conditions that must be satisfied before an operation, while postconditions generally state the changes caused by the operation. For example, the operation **put-on(block1, block2)**, which puts block1 on top of block2 in blocks world, has the following preconditions and postconditions:

Preconditions: there must be no other object on top of block2, and the top of block2 must be flat;
Postconditions: block1 is on-top-of block2; block1 is "deleted" from its previous position.

This example shows how a system can acquire the rules by learning from examples. In this example, the world consists of four objects: two cubes, a cylinder and a pyramid. The following variables are defined for SPARC/G:

(1) top-of-cube1: its value is the name of the object that is on top of cube1;

(2) top-of-cube2: its value is the name of the object that is on top of cube 2;

(3) top-of-cylinder: its value is the name of the object that is on top of the cylinder;

(4) top-of-pyramid: its value is the name of the object that is on top of the pyramid;

(5) put: the action of putting an object on top of another; for example [put = cylinder-on-cube1], specifies the action of putting the cylinder on top of cube1;

(6) put-on-cube1: a binary variable that states the legitimacy of putting an arbitrary object on top of cube1.

The positive events given to SPARC/G are arbitrary legitimate actions and statuses permitted by the blocks world. The negative events are, on the other hand, illegitimate actions and status. Here is one example given to the program:

[top-of-cube1 = cube2][top-of-cube2 = clear]
[top-of-cylinder = clear][put = cylinder-on-cube2][put-on-cube1 = no]

This example states that if cube2 is on top of cube1, and the tops of the cube2 and cylinder are clear, then one may put the cylinder on top of cube2 (put = cylinder-on-cube2) but may not put anything on cube1 (put-on-cube1 = no). The input episode is shown in Table 5.

Table 5 Input events for Example 2.

Event	top-of-cube1	top-of-cube2	top-of-cylinder	put	put-on-cube-1
1	clear	clear	clear	cube2-on-cube1	yes
2	clear	clear	clear	cylinder-on-cube1	yes
3	clear	clear	clear	pyramid-on-cube1	yes
4	clear	clear	clear	cube2-on-pyramid	no
5	clear	clear	clear	cube1-on-cylinder	yes
6	cube2	clear	clear	cylinder-on-cube2	no
7	cube2	clear	clear	cube2-on-cube1	no
8	pyramid	clear	clear	cylinder-on-cube1	no
9	cylinder	clear	clear	pyramid-on-cube1	no

SPARC/G discovered the following rule using the DNF model with a lookback of 0 in 2.2 seconds:

[top-of-cube1 = clear] v [put-on-cube1 = no]

which can be reexpressed as

[put-on-cube1 = yes] → [top-of-cube1 = clear]

which in effect says that if you want to put something on top of cube1 then the top of it must be clear. This is obviously correct. On the other hand, this example shows one of the limitations of SPARC/G: the currently used description language allows only one-argument functions or predicates. A desirable extension of the program would be to allow in its description language predicates and functions of two or more arguments.

Example 3: Learning a Symbolic Description of Motion

Motion is one of the most basic notions that governs our understanding of the physical world. How does motion of an object affect the state of the world and what type of motions are possible given the state of the world? The answer depends on discovering relations governing motion. We need not know Newtonian mechanics to understand the physical interactions of motion. The first step toward such discovery is to hypothesize causal connections between descriptions of the world. Since motion occurs in time, a sequential pattern recognition program like SPARC/G can play an important role. This example illustrates how the program can discover the causal relationships between the state of a spring and motion of an object. The program was given the following descriptors as perceptual vocabulary:

(1) Spring (state of the spring): {compressed, relaxed, stretched};

(2) Pos (the position of the block with respect to the position of the spring at rest): {left, center, right};

(3) Move (direction of the movement of the block): {left, still, right};

(4) Accel (the block slows down, accelerates, or moves with constant speed): {-1, $+1$, 0}.

Initially, the spring is stretched and the spring oscillates back and forth as shown in Figure 6. The corresponding input episode is shown in Table 6.

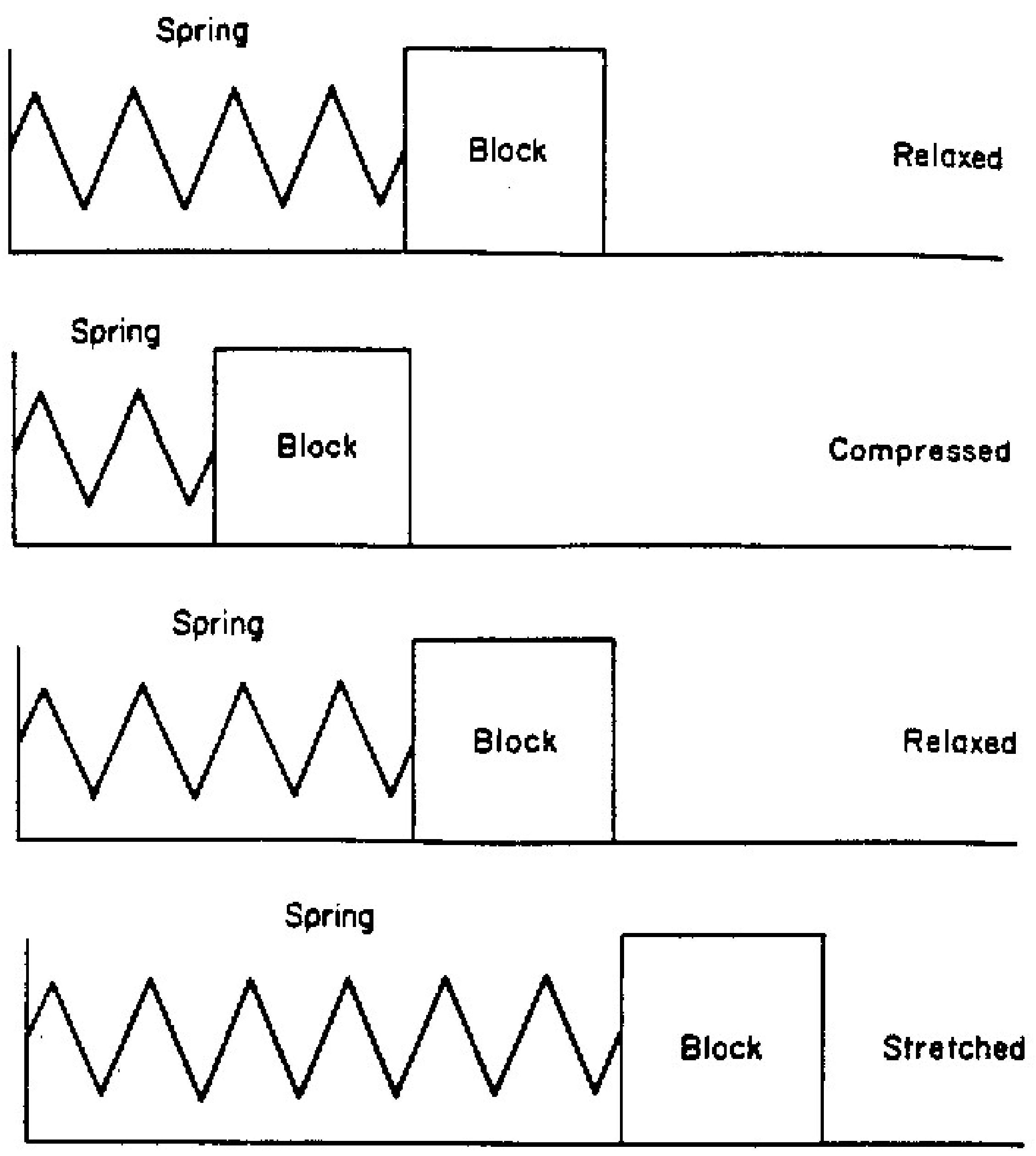

Figure 6 The oscillating block.

SPARC/G discovered the following decomposition rule with lookback of 1 in 1.1 seconds:

Rule 1: decomposition model, lookback: 1, nphases: 0
[Spring1 = stretched] → [Move0 = left]
[Spring1 = relaxed] → [Move0 = still]
[Spring1 = compressed] → [Move0 = right]

The rule can be paraphrased as follows:

(1) if the spring is stretched then the block is going to move to the left;

(2) if the spring is relaxed then the block is coming to a halt;

(3) if the spring is compressed then the block is going to move to the right;

SPARC/G was able to predict the movement of the block from the state of the spring. Even though this rule may fail (for example, if the spring is stretched too much then it may break), it seems to be a good first approximation of our intuitive notion of spring motion.

Table 6 Input events for Example 3.

Event number	Spring	Pos.	Move	Accel.
1	stretched	right	still	−1
2	relaxed	center	left	0
3	compressed	left	still	+1
4	relaxed	center	right	0
5	stretched	right	still	−1
6	relaxed	center	left	0
7	compressed	left	still	+1
8	relaxed	center	right	0
9	stretched	right	still	−1
10	relaxed	center	left	0
11	compressed	left	still	+1
12	relaxed	center	right	0
13	stretched	right	still	−1
14	relaxed	center	left	0
15	compressed	left	still	+1
16	relaxed	center	right	0
17	stretched	right	still	−1
18	relaxed	center	left	0
19	compressed	left	still	+1
20	relaxed	center	right	0
21	stretched	right	still	−1
22	relaxed	center	left	0
23	compressed	left	still	+1
24	relaxed	center	right	0
25	stretched	right	still	−1
26	relaxed	center	left	0
27	compressed	left	still	+1
28	relaxed	center	right	0
29	stretched	right	still	−1
30	relaxed	center	left	0
31	compressed	left	still	+1
32	relaxed	center	right	0
33	stretched	right	still	−1
34	relaxed	center	left	0
35	compressed	left	still	+1
36	relaxed	center	right	0

Physicists can explain the episode from first principles, but most human beings are not physicists. It seems that we typically derive qualitative relations existing in the world by doing inductive inferences from our observations, such as those performed by SPARC/G. Thus it appears that the program can be used to capture some important aspects underlying our processes of acquiring models of the physical world.

Example 4: Learning a Sequence of Actions

The operation of most planning or problem-solving systems is usually based upon a predefined set of rules. These rules represent the direct injection of knowledge from human users to the system. In this example, we show how SPARC/G can be used to acquire these rules by learning from training episodes.

Suppose we wish to teach a robot to operate a simplified cassette recorder by giving examples. Several legitimate actions are defined for the robot, such as to put a cassette into the recorder, eject the cassette, play, stop, etc. The robot is allowed to play with the cassette recorder, and a tutor labels each of the robot's actions as being either correct or incorrect. Whenever the robot effects an incorrect action, it is assumed that the robot will retract the incorrect action before making any further attempt. The robot must figure out the right sequence of actions all by itself. It is assumed that the rules to be learned are in the form

$$\text{ACTIONi} \rightarrow \text{ACTIONj or ACTIONk}$$

Such a rule states that after ACTIONi is executed, the next legitimate action can only be either ACTIONj or ACTIONk.

In this example, four legitimate actions on the recorder are defined:

(1) Put: putting the cassette into the recorder;

(2) Play: begin playing the cassette;

(3) Stop: stop playing the cassette;

(4) Eject: taking the cassette out of the recorder.

The legitimate sequence of actions is as follows:

(1) after putting the cassette into the recorder (Put), one may either eject the cassette (Eject) or start playing (Play);

(2) after begin playing the cassette (Play), the only legitimate action is Stop;

(3) after Stop, one may either Eject or Play;

(4) after Eject, the only legitimate action is Put.

A variable Action, among some other irrelevant variables, is defined in this example. The variable Action can take on either one of the four values: put, play, stop or eject.

Part of the episode given to SPARC/G is shown in Table 7.

Table 7 Input episode for Example 4.

Event number	Action	Legal?
1	put	yes
2	put	no
3	stop	no
4	play	yes
5	put	no
6	stop	yes
7	eject	yes
8	play	no
9	put	yes
10	eject	yes
11	put	yes
12	eject	yes

SPARC/G discovered the legal sequences of actions using the decompositional model with a lookback of 1 in 2.5 seconds. The rules produced take the form of implications:

Rule 1: decomposition model, lookback: 1, nphases: 0
[action1 = eject] → [action0 = put]
[action1 = stop] → [action0 = play v eject]
[action1 = play] → [action0 = stop]
[action1 = put] → [action0 = play v eject]

The rules can be paraphrased as follows:

Following Eject, the next action must be a Put;

Following Stop, the next action must be either a Play or an Eject;

Following Play, the next action must be a Stop;

Following Put, the next action must be either a Play or an Eject.

Thus these rules exactly characterize the legal actions.

Example 5: ELEUSIS: A Game of Scientific Discovery

This example shows the program's capability to discover rules in the card game Eleusis that models the process of scientific discovery (Gardner, 1977).† The game is played between a dealer and several players. Given a

† *The New Eleusis* is available from Robert Abbott at Box 1175, General Post Office, New York, NY 10001, USA.

sequence of cards that represent an instantiation of a qualitative prediction rule invented by the dealer (e.g. alternating colour of cards), the players are supposed to guess the secret rule invented by the dealer. In order to make the game more interesting, the dealer is penalized for inventing rules too difficult for any one to discover, or rules so simple that everyone can discover them. For the purpose of this example, it is assumed that SPARC/G poses as a player trying to figure out the rule governing the card sequence. The following is a simple Eleusis example designed to show the versatility of SPARC/G. A specialized version of the program, SPARC/E, has shown expert level performance in playing the game, and has beaten its human counterparts on many occasions.

The card sequence is given as a main line and a side line. The cards (read from left to right) in the main line represent positive instances that conform to the dealer's secret rule, and the cards in the sidelines represent negative instances that defy the rule:

Main line	JC	AD	QH	10S	QD	9H	QC	7H
Side line	KC	5S				4S		10D

The above layout of cards shows a card sequence of alternating faces, with Jack, Queen and King as face cards. The layout indicates that it is legitimate to play an Ace of diamonds (AD) following a Jack of clubs (JC), but not a King of clubs (KC), etc.

When given the above sequence, SPARC/G discovered the dealer's secret rule in three ways:

Rule 1: decomposition model, lookback: 1, nphases: 0
[face(card1) = false] → [face(card0) = true] v
[face(card1) = true] → [face(card0) = false]

Rule 2: periodic model, lookback: 1, nphases: 1
period([face(card0) <> face(card1)])

Rule 3: periodic model, lookback: 1, nphases:2
period([face(card0) = true], [face(card0) = false])

The rules can be paraphrased as follows.

Rule 1 If the previous card is a face card, then the next card must be a non-face card. If the previous card is a non-face card, then the next card must be a face card. This rule was discovered using the decomposition model with a lookback of one. (4 seconds)

Rule 2 Adjacent cards in the card sequence have different face values. This rule was discovered using the periodic model with a phase of one. (1 second)

Table 8 Results from other ELEUSIS game sessions.

Secret rule	Rule discovered	Execution time (s)	Source of the rule
• If previous card is red then play a faced card; If previous card is black then play a nonfaced card.	Rule 1: lookback: 1 nphases: 0 Decomposition [color(card1) = red] → [face(card0) = true] v [color(card1) = black] → [face(card0) = false]	2.9	Tom Channic
• If previous card is odd then play a card of different colour; If previous card is even then play a card of same colour.	Rule 1: lookback: 1 nphases: 0 Decomposition [parity(card1) = odd] → [color(card0) <> color(card1)] v [parity(card1) = even] → [color(card0) <> color(card1)]	1.6	Donald Michie
• Play any card that is either red and odd, or black and even.	Rule 1: lookback: 0 nphases: 0 DNF [color(card0) = red][parity(card0) = odd] v [color(card0) = black][parity(card0) = even]	1.2	Patrick Winston
If previous card is odd then play a black card; If previous card is even then play a red card.	rule 1: lookback: 1 nphases: 0 Decomposition [parity(card1) = odd] → [color(card0) <> color(card1)] v [parity(card1) = even] → [color(card0) <> color(card1)]	1.5	Gardner (1977)

Rule 3 The sequence is composed of two interleaving sequences of cards, where one sequence are all face cards, and the other sequence all non-face cards. This rule was discovered using the periodic model with a phase of two. (1 second)

Table 8 shows the result of several other game sessions.

5 SUMMARY AND RESEARCH DIRECTIONS

The methodology presented is applicable to a wide range of qualitative prediction problems. The major strengths of the methodology lie in its generality and the use of several description models and corresponding sequence transformations. These models and transformations guide the search through an immense space of plausible qualitative prediction rules. The representation space of DNF and Decomp Model is $2^{A\times(lb+1)\times v\times D}$, and $2^{A\times(lb+1)\times v\times p^R}$ for periodic model, where A is the number of attributes, lb is the lookback parameter, v is the size of the domain of an attribute, D is the maximum number of disjunctive terms, p is the number of phases in periodic model, and R is depth of recursion in periodic model.

The methodology assumes that the information contained in the events, plus the information that can be inferred from the events using the program's background knowledge, is sufficient to predict a plausible continuation of a process. One way to improve the capability of the system is to enhance the background knowledge and the program's ability to utilize this knowledge. The current implementation utilizes mainly the information contained in the events, and to a lesser extent those contained in the background knowledge.

Background knowledge consists primarily of description models and associated sequence transformations, domains and types of variables, and various domain-specific constructive induction rules that generate new variables from the old ones. It does not, however, have capabilities for testing the consistency of generalized selectors in the complexes for utilizing various interdomain constraints, or for performing a chain of deductions to see if the episode is explained by the rules of inference in the background knowledge (Dejong, 1986).

The search strategy invokes two processes simultaneously:

(1) a specialization of description models by instantiating the models with the given parameters to generate restricted rule forms;

(2) a transformation of the original episode into a new form, more amenable for rule discovery.

The algorithms presented here work best when negative events are available, but satisfactory performance can be obtained without negative events. Processes that contain noise or error are currently not handled by the program.

The generality of the program has been demonstrated by a series of examples from different domains. Among desirable paths for future research are improving the efficiency of the search process, extending the representation to more powerful description language such as the annotated predicate calculus (Michalski, 1983) so that multiple-argument descriptions are allowed, and developing the capability for incremental learning.

ACKNOWLEDGEMENTS

The authors are grateful to Peter Haddawy, Carl Kadie, Igor Mozetic, Gail Thornburg and Carl Uhrik for their comments on the earlier draft of this paper. This research was supported in part by the National Science Foundation under grant NSF DCR 84-06801, the Office of Naval Research under grant N00014-82-K-0186 and the Defense Advanced Research Projects Agency under grant N00014-K-85-0878.

REFERENCES

Box, G. E. P. and Jenkins, G. M. (1976). *Time-Series Analysis: Forecasting and Control*, Revised edn. San Francisco: Holden-Day.

Chilausky, R., Jacobsen, B. and Michalski, R. S. (1976). An application of variable valued logic to inductive learning of plant disease diagnostic rules. *Proc. 6th Ann. Symp. on Multiple Valued Logic, Logan, Utah, 1976.*

DeJong, G. (1986). An approach to learning from observations. *Machine Learning: An Artificial Intelligence Approach*, Vol. II. (ed. R. S. Michalski, J. G. Carbonell and T. Mitchell). Los Altos, California: Morgan Kaufmann.

De Kleer, J. and Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence* **24**, 7–83.

Dietterich, T. G. (1980). The methodology of knowledge layers for inducing description of sequentially ordered events. M.S. thesis, Department of Computer Science, University of Illinois, Urbana.

Dietterich, T. G. and Michalski, R. S. (1979). Learning and generalization of characteristic descriptions: evaluation criteria and comparative review of selected methods. *Proc. 6th Int. Joint Conf. on Artificial Intelligence, Tokyo, August 1979*, pp. 223–231.

Dietterich, T. G. and Michalski, R. S. (1985). Discovering patterns in sequences of events. *Artificial Intelligence* **25**, 187–232.

Forbus, K. (1984). Qualitative process theory. Ph.D. thesis, MIT.

Gardner, M. (1977). On playing the New Eleusis, the game that simulates the search for truth. *Scientific American* 237 (October), 18–25.

Hedrick, C. L. (1974). A computer program to learn production systems using a semantic net. Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh.

Hunt, E. B. (1966). *Experiments in Induction*. New York: Academic Press.

Larson, J. (1976). A multi-step formation of variable logic hypotheses. *Proc. 6th Int. Symp. on Multiple-Valued Logic, Logan, Utah, 1976.*

Larson. J. (1977). Inductive inference in the variable valued predicate logic system VL21: Methodology and computer implementation. Rep. 869, Dept Computer Sci. Univ. Illinois, Urbana.

Larson, J. and Michalski, R. S. (1977). Inductive inference of VL decision rules. *SIGART Newsletter* (June), 38–44.

Michalski, R. S. (1969). Algorithm Aq for the quasi-minimal solution of the covering problem. *Archiwum Automatyki i Telemechaniki*, No. 4, Polish Academy of Sciences. (In Polish.)

Michalski, R. S. (1972). A variable-valued logic system as applied to picture description and recognition. *Proc. IFIP Working Conf. on Graphic Languages, Vancouver.*

Michalski, R. S. (1973). Discovering classification rules using variable-valued logic system VL1. *Advance Papers of 3rd Int. Joint Conf. on Artificial Intelligence, Stanford University*, pp. 162–172.

Michalski, R. S. (1974). Variable-valued logic: System VL1. *1974 Int. Symp. on Multiple-Valued Logic, West Virginia University, Morgantown, West Virginia, 29–31 May.*

Michalski, R. S. (1977). Variable-valued logic and its application to pattern recognition and machine learning. *Computer Science and Multiple-Valued Logic* (ed. D. C. Rine), pp. 506–534. Amsterdam: North-Holland.

Michalski, R. S. (1980). Pattern recognition as knowledge-guided inductive inference, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 4, pp. 349–61, July 1980.

Michalski, R. S. (1983). A theory and methodology of inductive learning. *Machine Learning: An Artificial Intelligence Approach* (ed. R. S. Michalski, J. Carbonell and T. Mitchell), pp. 83–134. Palo Alto, California: TIOGA Publishing Co.

Michalski, R. S., Chen, K. and Ko, H. (1985). SPARC/E(V.2): A Eleusis rule generator and player. Rep. Dept Computer Sci., Univ. Illinois, Urbana.

Mitchell, T. M., Richard, M. K. and Kedar-Cabelli, S. T. (1985). Explanation-based generalization: A unifying view. Rutgers Computer Sci. Dept Tech. Rep. ML-TR-2.

Schwenzer, G. M. and Mitchell, T. M. (1977). Computer-assisted structure elucidation using automatically acquired carbon-13 NMR rules. *ACS Symp. Ser.* 54: *Computer-Assisted Structure Elucidation* (ed. D. H. Smith).

Soloway, E. and Riseman, E. M. (1977). Knowledge-directed learning. *Proc. Workshop on Pattern Directed Inference Systems. SIGART Newsletter* (June), 49–55.

Waterman, D. A. (1975). Serial pattern acquisition: A production system approach. Working Paper 286, Dept Psychology, Carnegie-Mellon Univ. Pittsburgh.