Journal of Applied Intelligence 4, 297–316 (1994)

Graph-Based Induction as a Unified Learning Framework

KENICHI YOSHIDA
Advanced Research Laboratory, Hitachi, Ltd., Hatoyama, Saitama 350-03, Japan
YOSHIDA@HARL.HITACHI.CO.JP

HIROSHI MOTODA
Advanced Research Laboratory, Hitachi, Ltd., Hatoyama, Saitama 350-03, Japan
MOTODA@HARL.HITACHI.CO.JP

NITIN INDURKHYA*
Advanced Research Laboratory, Hitachi, Ltd., Hatoyama, Saitama 350-03, Japan
NITIN@CS.SU.OZ.AU

Abstract. We describe a graph-based induction algorithm that extracts typical patterns from colored digraphs. The method is shown to be capable of solving a variety of learning problems by mapping the different learning problems into colored digraphs. The generality and scope of this method can be attributed to the expressiveness of the colored digraph representation, which allows a number of different learning problems to be solved by a single algorithm. We demonstrate the application of our method to two seemingly different learning tasks: inductive learning of classification rules, and learning macro rules for speeding up inference. We also show that the uniform treatment of these two learning tasks enables our method to solve complex learning problems such as the construction of hierarchical knowledge bases.

Key words: Machine learning, induction, graph

1. Introduction

In recent years, much of machine learning research has concentrated on algorithms for two relatively distinct learning tasks: 1) extracting new knowledge from sample data [1, 2], and 2) reorganizing existing knowledge based on various considerations such as efficiency, operationality, etc. [3].

While several algorithms are available for each of the two learning tasks, some applications require the ability to address both problems simultaneously. For instance, the task of hierarchical knowledge-base construction involves both kinds of learning. The inference process at lower levels must be reorganized into more efficient processes at higher levels in the hierarchy, and at the same time knowledge bases at higher levels of abstraction must be induced from sample data at lower levels. Similar problems abound in the field of robotics [4]. Addressing such applications with existing learning methods requires a careful decomposition of the problem such that each part can be solved by existing algorithms. Hence, such problems have been difficult to solve, and existing solutions rely on considerable manual interaction. For example, current methods for generating hierarchical knowledge bases depend on manual guidance [5, 6].

This article examines methods for solving complex learning problems that demand several modes of learning to be performed. We present a unified method for extracting new knowledge

*Present address: Basser Department of Computer Science, University of Sydney, NSW 2006, Australia.

from sample data as well as reorganizing given knowledge for efficient problem-solving. The method relies on the use of colored digraphs for representing learning problems. Different learning problems are mapped into colored digraphs in a manner such that solving the learning problem is achieved by extracting common patterns in the resulting graph. This enables the use of a *single* graph-based induction algorithm for solving a variety of learning problems. Specifically, our method can be used to 1) learn classification rules from sample data, 2) learn macro rules for speeding up inference, and 3) create a hierarchical knowledge base.

The rest of the article is organized as follows: section 2 examines related work, section 3 outlines the basic idea of graph-based induction, section 4 describes how different learning tasks can be mapped into colored digraphs, section 5 examines the algorithm for extracting typical patterns, section 6 presents experimental results, and section 7 summarizes the key points and discusses future issues.

2. Related Work

Learning strategies that involve the use of both inductive and deductive methods have been proposed before [7–11]. Most of these methods, however, used separate components for inductive learning and deductive learning, and focus instead on how to combine these two separate learning functions. When problems cannot be decomposed easily, such methods are difficult to use.

Colored digraphs have been used for representation of deductive learning problems [12]. However, the use of this representation to solve other learning tasks, especially problems that require a combination of learning methods, has not been attempted before.

Several algorithms are available for extracting substructures from graph or graphlike data structures [13–18]. However, most of them [13–16] cannot extract substructures from a single connected structure and are not applicable for certain kinds of learning such as knowledge-base abstraction. The pattern extraction procedure in SUBDUE [17, 18] is quite general, but can be computationally prohibitive for large problems.

3. Graph-based Induction

We employ the colored digraph for representing learning problems. In a colored digraph representation, each node has topological (edge) information and one or more colors attached to it. The colors are used to associate attributes to each node. Colored digraphs can be viewed as a generalization of the attribute-value representation used in conventional classification learning systems [1, 2].

The central intuition behind our graph-based induction method is as follows: a pattern that appears frequently enough in a colored digraph is worth paying attention to and may represent an important concept in the environment (which is implicitly embedded in the input graph). In other words, the repeated patterns in the input graph represent typical characteristics of the given environment. The extraction of patterns is based solely on finding repetitions of substructures within the input graph. The algorithm analyzes the input colored digraphs, and extracts sets of patterns (each set is called a *view*), in such a way that the patterns in the view can be used to contract the input graph. A key operation in this procedure is graph matching.

Standard graph-matching methods check the equivalence of two graphs by considering all possible edge combinations. In our algorithm, however, we adopt graph identity as our matching criterion. The incident edges are ordered, and the equivalence between the corresponding edges is examined. An important implication of this is that graph matching can be done in polynomial time. This restriction does not seem to limit the class of learning tasks that the algorithm can handle. Figure 1 illustrates pattern matching based on graph identity. Note that although there are three isomorphic subgraphs in the input graph, the algorithm finds only the subgraph in the upper box as a typical pattern, gives this a new identifier color (eight in this case), and contracts the graph such that only the two subgraphs *identical* to the typical pattern are contracted.

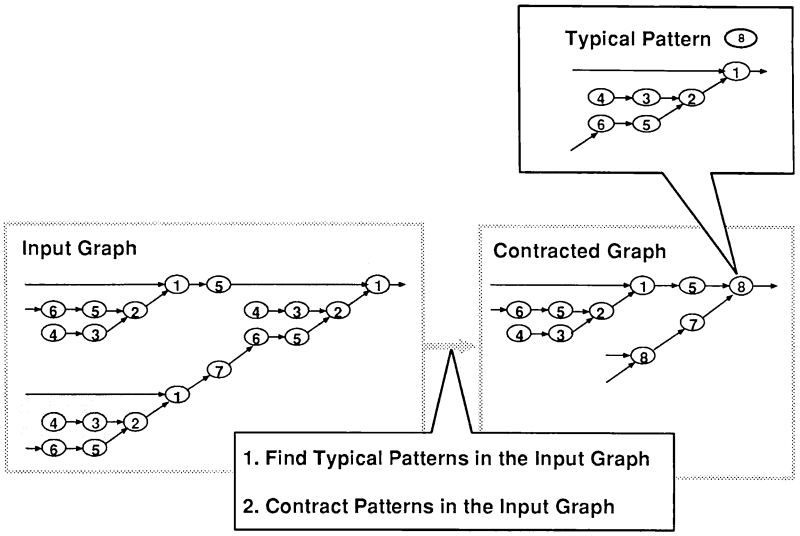


Fig. 1. Graph contraction by finding typical patterns.

Besides the graph contraction procedure, a method to evaluate the contracted graph is also required. This is done by *selection criteria,* based mainly on notions related to graph size, that enable selection of the best contracted graph. These issues are discussed in greater detail in section 5.

Interpretation of the typical patterns depends on the learning task encoded in the colored digraph. For example, in one interpretation, the typical patterns can be viewed as new knowledge induced from data encoded in the input graph. On the other hand, if the input graph represents inference patterns, then typical patterns may be interpreted as new inference macro-rules. Traditionally, these two learning tasks have been considered separately. However, by encoding them in colored digraphs, it is possible to solve both problems by a single algorithm. We now describe how different learning problems are encoded into the colored digraph representation.

4. Transforming Learning Problems into Colored Digraph Representation

In this section, we describe how to map different types of learning problems into colored digraphs. Besides the conventional learning problems such as inductive classification learning and macro-rule learning, we also illustrate how more complex learning problems that involve a combination of different learning strategies might be encoded in colored digraphs. It is important to note that the process of mapping learning problems into colored digraphs is done such that solution of the learning problem can be achieved by finding typical patterns in the resultant graph.

4.1. Conventional Learning Problems

In encoding conventional learning problems from inductive classification learning, speed-up learning, etc, we take advantage of the close connection between colored digraphs and the conventional representations used for these problems. For conventional inductive learning, the nodes represent the attributes of relevant entities, and node color is used to encode attribute values. For conventional deductive learning, the nodes in the graph represent the data referred to or produced by the inference engine. Two colors are used: one to encode the rule used to obtain the data and the other to encode the value or kind of data.

4.1.1. Learning Classification Rules from Sample Data. The process of mapping inductive learning problems into colored digraphs is best illustrated by an example.

We use the example of learning rules for classifying DNA promoter sequences. Figure 2 illustrates the complete process of mapping the problem into a colored digraph and then using the graph-based induction procedure to extract patterns and interpet them as classification rules. In this section, we focus on the first step only: mapping the DNA sequence data into colored digraphs.

In mapping the set of cases into the graph structure, we construct one subgraph for each sequence in the set of cases. The subgraph consists of a root node and a set of leaf nodes. The number of leaf nodes equals the number of attributes (in this case, the number of sequence elements). The color of the root node of the subgraph specifies whether the corresponding sequence represents a promoter sequence or not. The color of the *i*th leaf specifies the nucleotide (one of A, T, C, or G) of the *i*th position.

We now show that the learning problem is also transformed such that its resolution is equivalent to finding typical patterns in the colored digraph. The algorithm, briefly described in section 3, extracts patterns from the input graph and tries to decrease the contracted graph size. These typical patterns will consist of a root node and one or more leaf nodes. Note that each extracted pattern is equivalent to a typical DNA sequence (promoter or nonpromoter). In order to select "good" typical patterns, we need to use some selection criterion that reflects this interpretation of the typical patterns extracted. One such criterion is

$$\text{minimize} \left[\begin{array}{r} (\text{Number of Nodes in the} \\ \text{Contracted Graph}) \\ + (\text{Number of the different values of Color} \\ \text{in the Contracted Graph})^2 \end{array} \right] \quad [1]$$

The first term of the criterion is intended to encourage a search for typical patterns of the DNA data. The smaller the number of nodes in the contracted graph, the better the typical patterns used. The typical patterns extracted can be interpreted as classification rules. The second term in the selection criterion is a penalty to avoid an excessive number of rules and ensures that compact solutions are preferred.

This process ensures that the means of extracting typical patterns is equivalent to the goal of extracting classification rules. In appendix A, we discuss alternative encoding methods, selection criteria, and other related issues.

Note that the restriction on graph matching (see section 3) is necessary to map the learning problem correctly. Checking for equivalence between all possible edge combinations would be

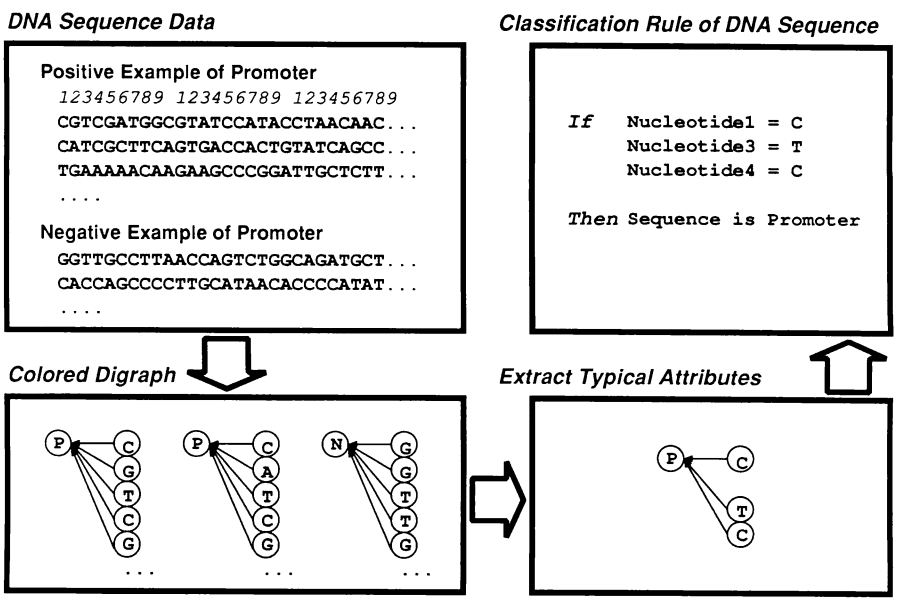


Fig. 2. Extraction of classification rules from DNA sequence data.

incorrect, since doing so would involve comparing different attributes.

The mapping process described above for the DNA sequence example is applicable for learning of all attribute-value type classification systems and is summarized in figure 3.

4.1.2. Learning Macro Rules for Speeding up Inference. Learning macro rules to make problem-solving more efficient is also a well-studied problem in machine learning. This involves reorganizing prespecified knowledge into a more efficient form. The complete procedure for the specific problem of equation-solving is illustrated in figure 4.

The colored digraph for this purpose is a type of proof tree. Each node corresponds to a term in the proof process. Each node has two colors. Color1 of a node corresponds to the axiom (or rule) used to prove the term and Color2 refers to the term itself. Here, Color2 is ignored in the matching process, and the only equivalence for Color1 is examined. The ordering of edges from a node encodes the ordering of terms in the proof. For example, in the case of a prolog clause, the ordering of edges from a node indicates the order in the body part of the clause.

The learning problem (finding macro rules to make equation-solving more efficient) is consistent with the goal of finding typical patterns. These patterns can be interpreted as macro rules. This interpretation process for the macro rules is essentially equivalent to the generalization process of Explanation-Based Learning (EBL) [19, 20]. In order to select "good" typical patterns, the following selection criterion can be used:

$$\text{minimize} \left[\begin{array}{r} (\text{Number of Nodes in the} \\ \text{Contracted Graph}) \\ +(\text{Number of different Color1 values} \\ \text{in the Contracted Graph})^2 \end{array} \right] \quad [2]$$

This criterion is very similar to that used for solving inductive learning problems. Similar to EBL, the goal here is to make inference efficient by chunking out the intermediate inference. The first term is intended to estimate the effect of chunking. The second term is a penalty for the excessive macro rules. The degradation of inference efficiency due to excessive number of macro rules is widely recognized in EBL research. The second term is intended to avoid such degradation.

The mapping process described above for equation-solving is applicable for other macro-rule learning problems of the type discussed in [21, 22] and is summarized in figure 5.

4.2. Hierarchical Knowledge-Base Construction

In this section we illustrate how more complex learning problems can be mapped into colored digraphs. We examine the problem of hierarchical

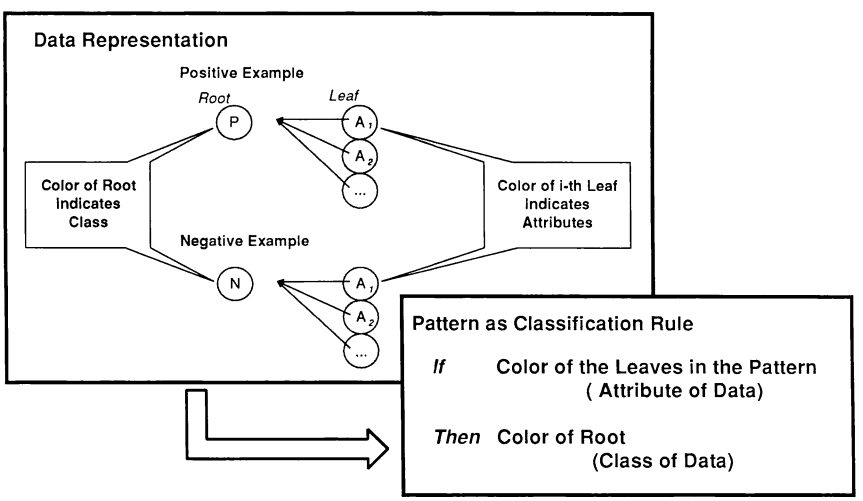


Fig. 3. Graph representation for classification rule learning.

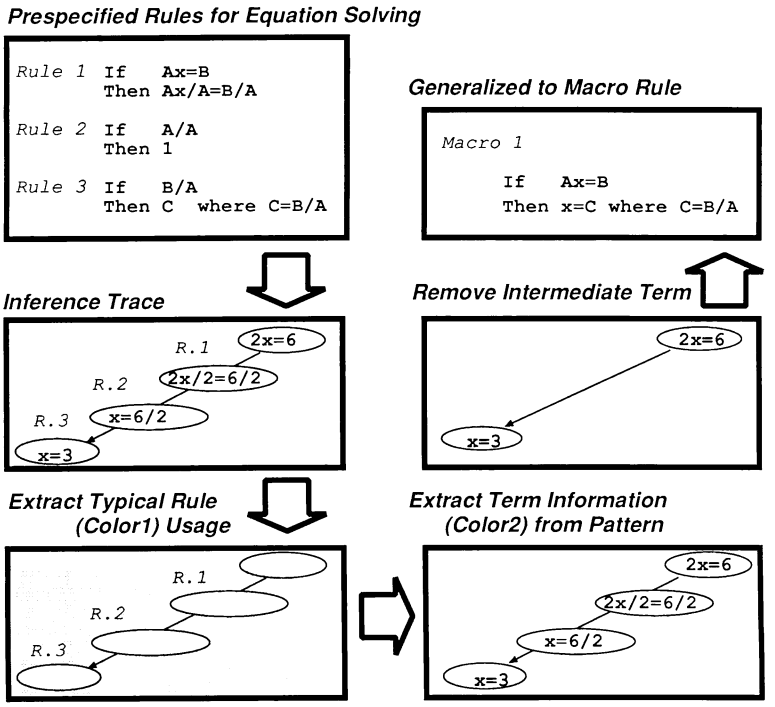


Fig. 4. Learning macro rules for equation solving.

knowledge-base construction and show how it can be solved by means of graph-based induction. The solution of this problem calls for a mixture of inductive and deductive learning strategies. Before we discuss how to map this problem into colored digraphs, we briefly describe hierarchical knowledge bases.

4.2.1. Hierarchical Knowledge Bases. A hierarchical knowledge base is a method of organizing knowledge about complex systems. Usually the different levels in the hierarchy are abstraction levels. Each abstraction level contains knowledge of the system appropriate at that level. Usually each level has its own *interpreta-*

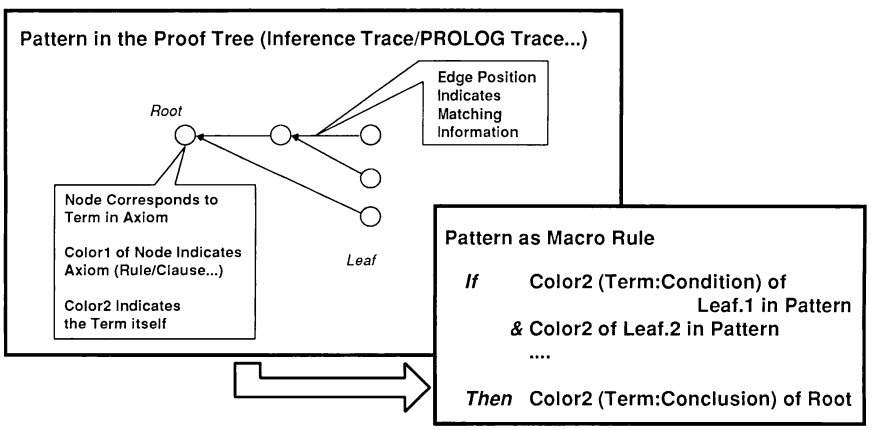


Fig. 5. Graph representation for macro rule learning.

tion rules for manipulating the objects at that level and making inferences about them. The levels are connected together by *reformation rules* that specify how to map knowledge at one level to the level immediately above (or below) it.

Figure 6 illustrates a two-level hierarchical kowledge base for describing an electrical circuit. At the lower level, the circuit is described in terms of node voltages and line currents, while at the higher level it is described in terms of logic gates and transistors. At each level, knowledge about the circuit is organized using an entity-relationship model. An example of a relationship between entities at the lower level is $I_2 = I_2 + I_3$, which describes the relationship between three line currents in the circuit. Various interpretation rules are available at each level that enable inferences to be made. These inferences can be used, for instance, to perform a qualitative simulation of the circuit at that level. The following is an example of an interpretation rule that describes additive relationships at the lower level of the knowledge base:

If $I_1 = I_2 + I_3$ **and** $I_2 = [+]$ **and** $I_3 = [+]$
Then $I_1 = [+]$

Besides the interpretation rules, the hierarchical knowledge base also consists of various reformation rules for describing the entities at the higher level (gates and transistors) in terms of entities at the lower level (node voltages and line currents).

Given such a hierarchical knowledge base, it is normally used in one of two modes:

Reformation Mode. Given a description at one level, descriptions at other levels are generated. For example, if the input description specifies relationships at the lower level of figure 6, the reformation rules are used to obtain logical relationships among the circuit elements at the higher level.

Inference Mode. In using the knowledge base to make inferences, an appropriate level is selected based on some criterion. Then the descriptions and interpretation rules at that level are used to make inferences.

Construction of such hierarchical knowledge bases is motivated by the fact that the higher levels enable more efficient inferences. The results of these inferences can then be mapped down to the lower levels if needed. This process is consid-

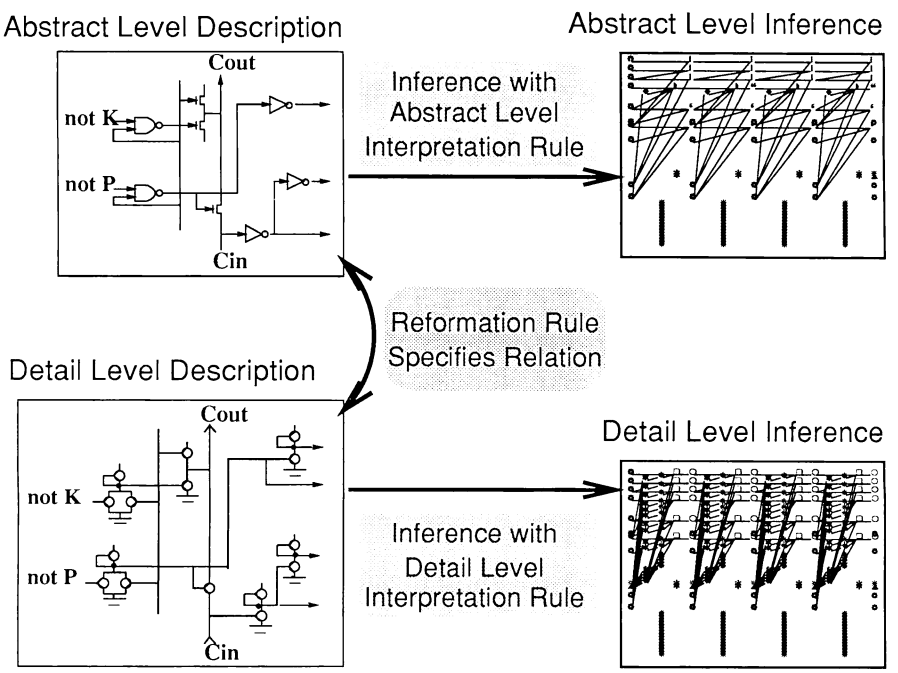


Fig. 6. A hierarchical knowledge base.

erably more efficient than performing the complete inference at the lower level itself.

Creation of hierarchical knowledge bases usually involves creating new abstract-level concepts (along with the accompanying interpretation rules for that level) from the lowest-level description and rules. Previous approaches to this problem [5, 6] have relied on user-supplied reformation rules.

In the next section, we discuss how to encode the problem into colored digraphs, thereby enabling the use of graph-based induction methods to learn both the interpretation rules as well as the reformation rules for each level.

4.2.2. Construction of Hierarchical Knowledge Bases by Colored Digraphs. In order to create hierarchical knowledge bases, qualitative simulation of the lowest level of the knowledge base is used. Figure 7 describes the overall process of mapping this information into colored digraphs and then using the graph-based induction procedure to extract patterns that can be interpreted as reformation and interpretation rules. In this section we focus on the first step: encoding the problem as a colored digraph.

The interpretation rules at the lowest level are used to obtain a qualitative simulation trace. It is this trace that is represented as a colored digraph. Each node corresponds to some physical datum, such as voltage and current at a certain node in the circuit. Two colors are used for each node: Color1 of the node corresponds to the interpretation rule used to calculate the value, and Color2 is the value itself. Each edge of a node corresponds to a variable in the interpretation rule associated with the node. Color2 is ignored in the matching process, but is used later on for extracting interpretation rules from the typical patterns.

We now show how finding typical patterns in the colored digraph is equivalent to constructing the hierarchical kowledge base. The typical patterns are interpreted in two ways: 1) they are viewed as reformation rules that translate the lower-level descriptions into higher-level vocabularies, and 2) they are also used to specify interpretation rules for the higher-level vocabularies. The process of extracting reformation and interpretation rules from typical patterns is shown in figure 7. Note that color2 values in the typical patterns are used in extracting the interpretation rules.

Thus, the process of finding typical patterns in the colored digraph results in automatic generation of the higher levels of the knowledge base. The contracted graph itself serves as the higher-level description. In order to obtain

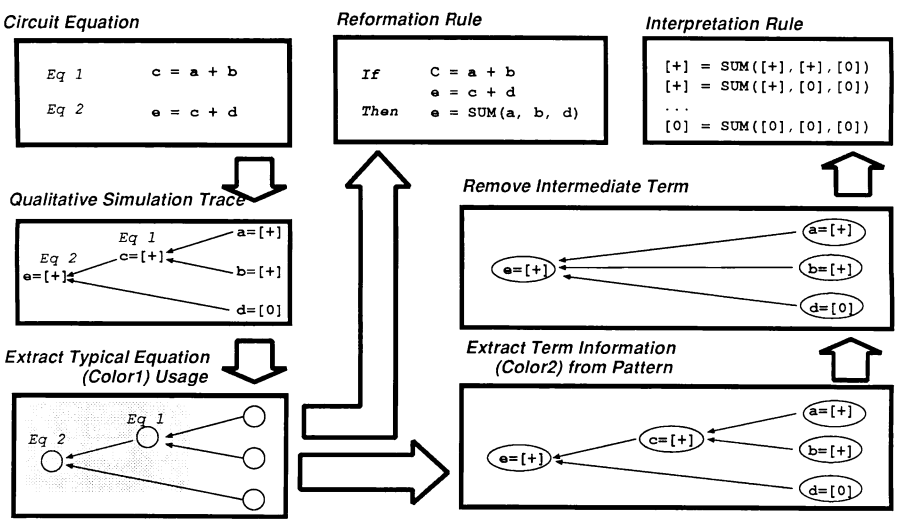


Fig. 7. Circuit equation reformation.

"good" higher levels, the following selection criterion was found useful:

$$\text{minimize} \left[\begin{array}{l} (\text{Number of Nodes in the Contracted Graph}) \\ + (\text{Number of Edges in the Contracted Graph}) \\ + \Sigma(\text{Number of Nodes in the Pattern}_j)^2 \end{array} \right] \quad [3]$$

The first term represents the amount of data to be handled, the second term the matching cost during inference, and the third term the cost to generate hierarchical descriptions from the input descriptions.

The mapping process illustrated above is applicable for other problems that involve creation of hierarchical knowledge bases based on QSIM [23] type qualitative simulators and is summarized in figure 8.

5. An Algorithm for Finding Typical Patterns

After the learning problem is encoded as a colored digraph, the key step is to extract typical patterns. In this section we describe in detail an algorithm, called CLiP, for performing this task. The algorithm is outlined in figure 9. CLiP is a beam-search algorithm, amenable to parallel implementation, that searches for typical subpatterns in the colored digraph. The objective is to find typical patterns that help contract the graph. In order to assist in the choice of desirable typical patterns, a selection criterion for comparing alternative contracted graphs is provided to the CLiP algorithm. The search procedure focuses on obtaining a reasonably good solution, not necessarily an optimal one. The method involves three basic operations: *Pattern Modification, Pattern Combination,* and *View Selection.* It starts with a set of null patterns, one for each view (a view holds promising typical patterns), and iteratively extends these patatterns by the first two operations (the old patterns are also retained). In each iteration, the input graph is contracted using patterns in each view, and only the good views are retained. The heart of the CLiP procedure involves iteratively performing the following three steps:

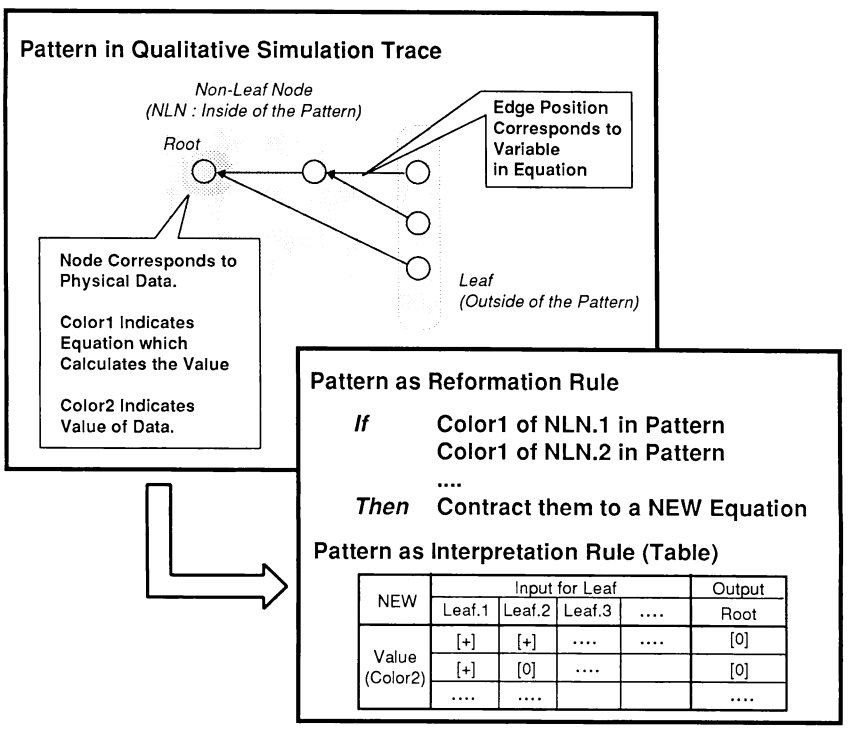


Fig. 8. Graph representation for hierarchical knowledge-base construction.

```
Algorithm CLiP(G_in, C, L, W)
    Input       G_in        : Colored Directed Graph
                C           : Selection Criterion
                L, W        : integer
    Output      Sequence of V_i where each V_i is
                                a Set of Typical Patterns in G_in
    Variable    B, B_next   : Set of Views
    begin
        V_0 ← ∅ ;     B ← {V_0} ;     i ← 1
        repeat L do
            B_next ← ∅
            for each V_tmp ∈ B do
                Call Pattern Modification
            Call Pattern Combination
            Call View Selection
            V_i ← Best view in B_next according to C
            i ← i + 1
        return Sequence of V_i
    end

Procedure Pattern Modification
    begin
        G_tmp ← Graph that is contracted from G_in
                                according to the patterns in V_tmp
        for each Temporary Pattern P in G_tmp do
            V_new ← V_tmp ∪ {Original Pattern of P}
            Append V_new to B_next
    end

Procedure Pattern Combination
    begin
        for each V_tmp1 ∈ B do
            for each V_tmp2 ∈ B do
                if V_tmp1 ≠ V_tmp2 then
                    V_new ← V_tmp1 ∪ V_tmp2
                    Append V_new to B_next
    end

Procedure View Selection
    begin
        B ← Top W views in B_next according to C
    end
```

Fig. 9. Algorithm for extracting typical patterns.

1. In pattern modification, each pattern in each view is extended. Figure 10 shows how patterns are modified in this step. First, a view is selected and the graph is contracted using patterns in the selected view. The reduced graph is then analyzed, and every possible pattern made up of two linked nodes is considered. In figure 9, these patterns are referred to as temporary patterns. Each such temporary pattern is then expanded based on patterns in the current view and is used to create new views. In the example in figure 10, three new views, each with two patterns, are generated from the current view. Note that patterns in the parent view are also stored in the new views.
   As mentioned earlier, the matching procedure is restricted to checking for graph identity. Standard graph matching checks the equivalence of two graphs by checking for graph isomorphism (i.e., all possible edge combinations). The matching procedure used in our study does not check for graph isomorphism and instead checks for graph identity. In contrast to matching based on graph isomorphism (an NP-complete problem), matching based on graph identity has a time complexity that is O(Number of Nodes).
2. In pattern combination, existing views are combined in pairs to obtain new views. All possible combinations are considered.
3. In view selection, estimates are obtained for each view as to how much reduction in the size of the graph can be expected after contracting the input graph using patterns in that view, and only those views that rank high are chosen within the allowable number of views.

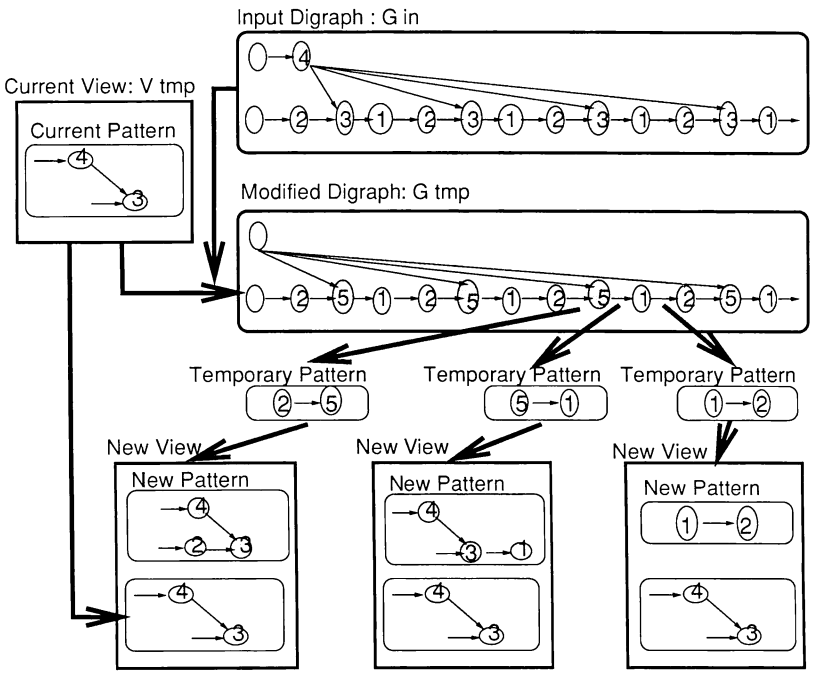


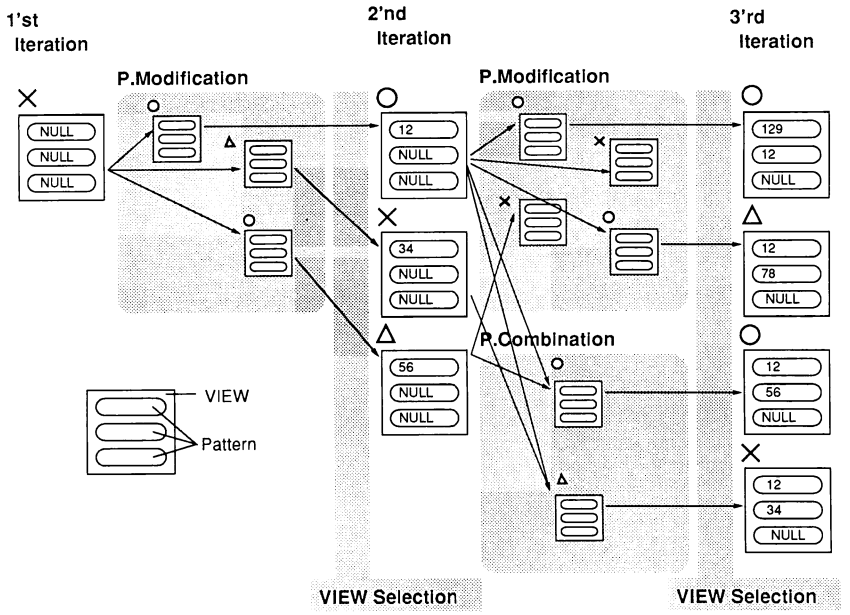
Fig. 10. Pattern modification.

Figure 11 illustrates how patterns evolve through generations. In this example, the maximum number of views is limited to four. In the first generation, starting from null patterns, the pattern-modification step generates three views, each containing one pattern that consists of two nodes (e.g., 1–2). In succeeding generations, besides pattern modification, pattern combintation is also performed. In pattern modification, all patterns in all views are considered in turn as candidates for modification. In each case, a new view is created consisting of the modified pattern appended to the original view. View selection is done after the pattern modification and combination steps. The view-selection step selects the best N (maximun number of views, which is a search parameter; four in figure 11) views in each generation. View selection is based on estimates of the relative effectiveness of views in graph contraction. These estimates are computed as follows: For each of the views $V_{Previous}$ (that were selected in the previous generation), the actual size $C_{Exact}(V_{Previous})$ of the contracted graph that results from applying patterns in that view to the graph is calculated at the beginning of the current generation. For each new view $V_{Current}$ that is generated by pattern modification step, the estimated size $C_{Estimate}(V_{Current})$ of the contracted graph that would result from applying that view is calculated as a perturbation to the actual size of the contracted graph due to the parent view $V_{Previous}$ (see equation (4)). The graph size for views generated by the pattern-combination step is estimated as the average of the two views involved in the combination.

The estimated size and the actual size (calculated at the beginning of the next generation) may not agree (see the difference between big and small marks on the top left of each view in figure 11). However, the estimate is reasonable enough to ensure that good views are usually selected.[1] The estimation process is necessary because computation of the actual size is computationally expensive.

$$C_{Estimate}(V_{Current}) = (1.0 - 0.1 \times F) \times C_{Exact}(V_{Previous}) \quad [4]$$

$$F = \frac{\text{No. of occurrences of Temp. Pattern}(A)\rightarrow(B)}{\text{No. of occurrences of Node}(B)}$$



The mark on the top left of each view indicates the result of graph contraction: ○ for good, × for bad and △ for intermediate. The size of a mark indicates whether the result is based on actual rewriting of the graph (big) or on estimation.

Fig. 11. Evolution of patterns in a view.

Note that the actual graph rewriting is performed only during the pattern-modification step (see figure 10) and is limited to the best N views. All necessary information for estimating graph size of other views is obtained in the process.

Our procedure for extracting typical patterns is closely related to SUBDUE [17, 18]. The key difference is that we use a more efficient graph-matching procedure.

6. Experimental Results

6.1. DNA Sequence Classification

A set of DNA promoter sequence data [24] was prepared for analyses by graph-based induction. The dataset consisted of 106 sequences, half of which were promoter sequences and the rest of which were nonpromoter sequences. Each sequence had 57 DNA nucleotides (one of A, T, C, or G). These data were encoded in a colored digraph as described earlier, and the CLiP algorithm was used to extract typical patterns. Since each generation creates views that give rise to classification rules, another issue is the selection of views from one particular generation as the final answer. A very good measure of success is the error rate for unseen data. Hence, those views that generate classification rules with minimum leave-one-out error [25] were selected in each generation. This technique was also used in [26] to select the final decision tree. The following classification rules were obtained:

$$F_{06} = A \wedge F_{15} = T \wedge F_{16} = T \rightarrow \textit{positive}$$
$$F_{16} = T \wedge F_{17} = G \rightarrow \textit{positive}$$
$$F_{15} = T \wedge F_{16} = T \rightarrow \textit{positive}$$
$$\textit{True} \rightarrow \textit{negative}$$

Table 1 summarizes the predictive performance of these rules and compares it to previous results. Error rates were estimated by the leave-one-out method. In table 1, ID3 [2] and SWAP1 [27] represent rule-learning systems, and BP a standard back-propagation neural network method with one hidden layer. CLiP outperforms the standard ID3 tree-induction program.

Table 1. Inductive learning: comparison with other classification methods

	Previously reported methods			
Method	ID3	SWAP1	BP	CLiP
Error/106	19	14	8	14

6.2. *First-Order Equation Solving*

Eighty-five inference traces of first-order equation solving were obtained by running a prolog program in which each prolog clause represented an axiom for equation solving. These traces[2] were mapped to colored digraphs as described earlier. The CLiP algorithm extracted a set of macro operators for more efficient equation solving. The macro operators were encoded as prolog causes, each representing a new theorem for equation solving. Figure 12 illustrates an inference trace before and after learning.

The effectiveness of the learning process can be assessed by computing the *speed-up ratio*, which compares the length of inferences before and after learning. The speed-up ratio was computed by adding the generated clauses at the top of the original prolog program and solving the equations with this new set of rules.

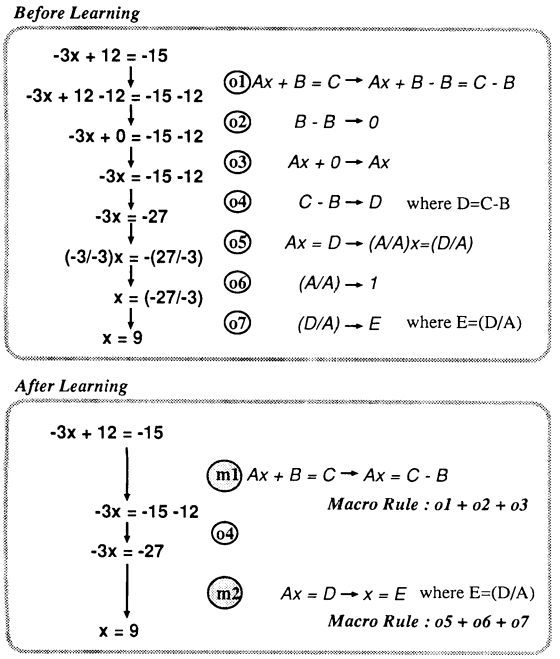


Fig. 12. Use of learned macro operators in equation solving.

As in the previous task, we need a measure to select a set of views as the final answer. The best measure of success here is the utility of acquired macros. So we select the views that show the best speed-up ability over the training problems. The results (table 2) indicate, in contrast to a nonselective EBL system, that learning by graph-based induction improved problem-solving efficiency.

6.3. *Circuit Equation Reformation*

Qualitative simulation results of an NMOS circuit that calculates a carry in a CPU [29] were used for constructing a hierarchical knowledge base of the circuit. The simulation traces were mapped to colored digraphs, and the graph-based induction algorithm successfully extracted the patterns and truth tables corresponding to NOR and NOT functions.

We set the maximum number of views at 15 and limited the algorithm to 50 iterations. The minimum graph size was attained in the 25th generation.

Unlike the previous two tasks, there is no uniform criterion to measure the usefulness of acquired concepts, and we select views that give the smallest contracted graph. Figure 13 shows an example of a typical pattern extracted (lower left) and the corresponding reformation/interpretation rules (right). The variables on the right-hand side of each equation in the conditional portion of the extracted concept (e.g., V and dV in equation①) are arranged so that they correspond to the numbers indicated on the edges pointing to each node (e.g., the edges 1 and 2 going into the node①). The reformation rule says that if there is a set of six relations (equations) as shown, it is reasonable to infer that V_{next} can be calculated by

Table 2. Equation solving: comparison with EBL

Method	Nonlearning	EBL	CLiP
CPU sec.	100.0	279.9	87.5
Speed-up Ratio	1.00	2.80	0.88

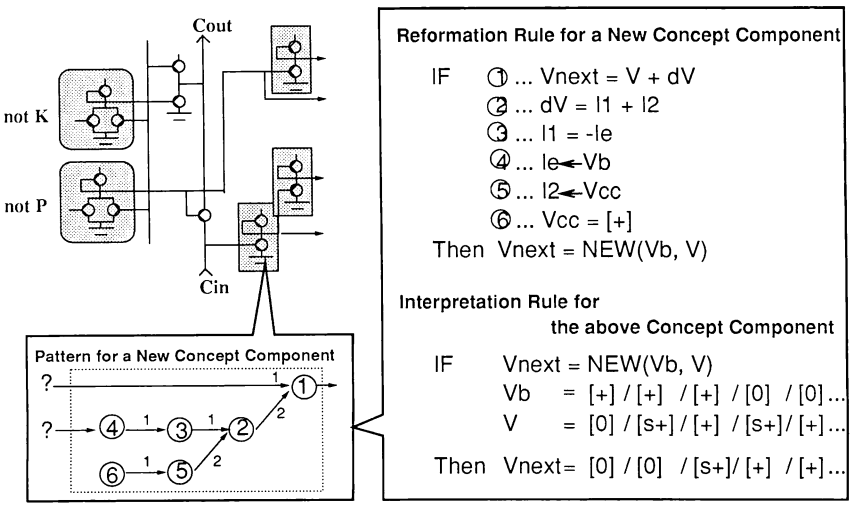


Fig. 13. A pattern that corresponds to NOT operation.

some relation *NEW* from the current value of V_b and V. The interpretation rule shown describes a set of relationships between the input V_b, V, and the output V_{next}. These relationships indicate that the concept generated is the analog NOT operation.

Further experiments were performed using different selection criteria. The graph-contraction process of one such experiment is shown in figure 14.[3] In this experiment, the third term of the selection criterion (the cost to generate hierarchical descriptions; see equation (5)), is multiplied by a factor of 10.

$$\text{minimize} \left[\begin{array}{l} \text{(Number of Nodes in the Contracted Graph)} \\ + \text{ (Number of Edges in the Contracted Graph)} \\ + \text{ 10 * (Number of Nodes in the Pattern}_j)^2 \end{array} \right] \quad [5]$$

The minimum size was attained at the 15th iteration, and the result corresponds to intermediate physical structures, i.e., pullup transistor and pulldown transistor. Using this level as input, the algorithm obtained the rightmost graph of figure 14, which is exactly the same as in the previous experiment.

These results indicate that the resultant structure varies depending on the characteristics of the inference system. While an inference system with strong reformation capability can recognize a complex object by one level of abstraction, another inference system with weaker reformation capability needs to levels of abstraction for the same object. In contrast to previous studies on hierarchical knowledge representation, our method automatically considers a number of factors (such as cost of inference) during the construction of hierarchical device representations. It is also able to automatically find new abstract-level concepts while at the same time recompiling the lower-level information into higher-level knowledge structures that involve the abstract concepts.

Table 3 shows the list of concepts generated.[4] All of these concepts are understandable to human experts, and some of them correspond to known mathematical and logical concepts. The lowest-level initial description for the circuit involves 112 equations. The generated concept *Carry Chain* in table 3 describes the overall behavior of the same circuit.

Achieving equivalent results solely by using conventional deductive learning techniques is difficult. The appropriate operationality criteria and goal concepts to cover the whole range of concepts listed in table 3 are difficult to specify. Similarly, conventional inductive-learning techniques are also not very useful for this task, since they lack the ability to reorganize the circuit equations. Note that all concepts listed in table 3

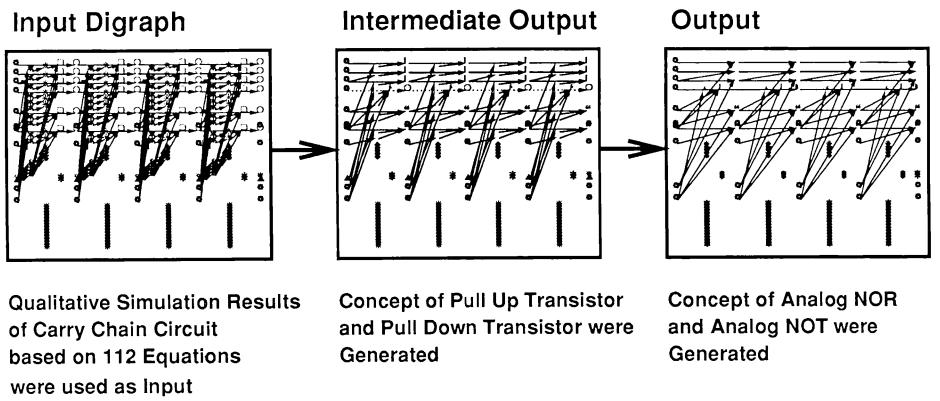


Fig. 14. Graph contraction process.

are reorganized from prespecified circuit equations.

6.4. Computational Considerations

In principle, the CLiP algorithm is amenable to parallel implementation. However, even on conventional hardware, computational requirements of the algorithm are modest. The most compute-intensive parts of the procedure involve graph matching and graph contraction. By using an efficient graph-matching procedure, and by using estimators to avoid having to do graph contraction too often, both these stages are well optimized.

For the hierarchical knowledge-base creation task, the input graph had 2176 nodes and 2144 edges. Using this graph, the concepts of Analog NOT/NOR were obtained in about six minutes of CPU time on a SPARCstation-1. The largest graph we have considered to date involved over 50K nodes and over 50K links, representing an inductive-learning problem involving data from the protein identification resource (PIR) [30]. Solving this problem required about two days of CPU time on a SPARCstation-1.

7. Discussion

In this article, we have examined the use of colored digraphs in integrating different modes of learning. We have shown how different learning problems can be mapped into colored digraphs. The mapping is achieved such that resolution of the corresponding learning task can be achieved by finding typical patterns in the resultant colored digraph. The use of a common representa-

Table 3. List of generated concepts in the hierarchical knowledge base

No.	Generated Concept	Comment
1.	Pull Up Transistor	Circuit made up of pure transistor, capacitor, and power source
2.	Pull Down Transistor	Circuit made up of pure transistor, capacitor, and ground
3.	Analog NOT	Circuit made up of pull up transistor and pull down transistor Inference table contains analog element
4.	Analog NOR	Circuit made up of pull up transistor and 2 pull down transistors Inference table contains analog element
5.	Digital NOT	Similar to Analog NOT Inference table does not contain analog element
6.	Digital NOR	Similar to Analog NOR Inference table does not contain analog element
7.	Carry Chain	Circuit which calculates carry

tion allows us to conceive of a *single* algorithm to perform a variety of inductive and deductive learning tasks such as learning classification rules, generating new macro operators, and creating hierarchical knowledge bases by identifying new concepts. The algorithm for extracting typical patterns relies on an efficient matching procedure. The interpretation of the typical patterns depends on the learning task being solved. For classification learning, they are interpreted as classification rules; in speed-up learning, they are viewed as macro operators; and in hierarchical knowledge-base construction, typical patterns help obtain both reformation rules as well as interpretation rules by identifying new concepts.

Empirical results indicate the viability of our graph-based induction method for solving a variety of learning problems. While it is reasonable to expect that methods for solving a specific learning problem might do better in certain specialized representations, our ultimate goal is to devise a unified learning mechanism, and towards this end the colored digraph representation is more promising. However, to accomplish our ultimate goal, further research is necessary. For example, we have not addressed the issue of defining selection criteria in a principled fashion. We still need to select such criteria carefully for each task. Methods to convert other learning problems into colored digraph representation still have room for investigation. For example, although the application to the PRODIGY [31] type control rule learning appears to be straightforward, experimental results have not yet been analyzed. We currently use acyclic graph to represent data and do not use continuous value. Further investigation is necessary on these points.

Acknowledgments

We would like to thank Sholom Weiss, Haym Hirsh, Pat Langley, and Hari Narayanan for helpful comments on the research. We would also like to thank, Ayumi Shinohara for providing the PIR data.

Appendix A. Representations for the Inductive Learning

The representation presented in section 4.1.1 seems to work for other inductive tasks. However, it has two problems:

- Negative conditions as as "*If the i-th nucleotide is NOT A*" cannot be represented.
- Using this representation, the algorithm may erroneously classify two patterns, each of which consists of the same nucleotides occurring in the same order—one as positive and the other as negative. This error occurs because this representation does not have a notion of mutual exclusiveness.

The solution of the first problem involves the use of a binary representation for the attributes, as shown in figure 15 for the DNA data. The root nodes have the same information as before (information about class), but the number of leaf nodes increases by a factor equal to the number of categories over all attributes. For the DNA data, for example, the number of leaf nodes increases by a factor of four. Each of these leaf nodes corresponds to a boolean attribute, indi-

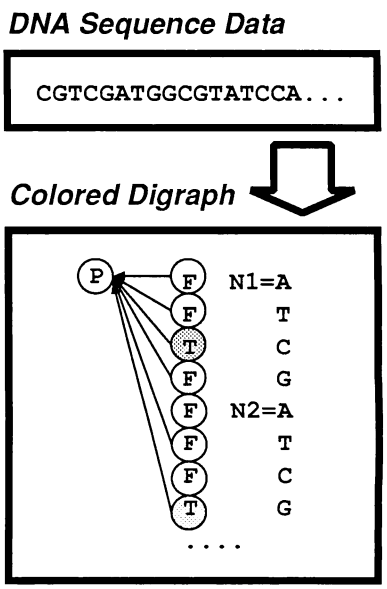


Fig. 15. Alternative representation for inductive learning.

cating the presence or absence of the relevant category for that attribute.

The second problem can be resolved by using the color information of the root node only in the selection process, along with the following selection criterion:

$$\text{minimize}\left[\begin{array}{l}(\text{Number of Nodes in the Contracted Graph}) \\ +(\Sigma \textit{Penalty}(\text{Pattern}_j)^2 \\ \Sigma \textit{Penalty}(\text{Pattern}_j) = (\text{Number of different root node colors in the set of patterns})\end{array}\right] \quad [6]$$

The second term in the selection criterion is a penalty for patterns that violate the mutual exclusiveness constraint. The utility of this term is illustrated in figure 16. With two classes, the number of the different colors for the root node of patterns that violate the mutual exclusivity constraint is two (positive and negative), and thus the value of *Penalty* is four. In contrast, for patterns that satisfy the constraint, the value of *Penalty* is one. Note that the class information is ignored during the matching process, but is used in the selection process.

The selection criterion presented here is also applicable to PRODIGY-type control rule learning [31], since PRODIGY's control rules for *SUCCEEDS*, *FAILS*, etc., are mutually exclusive concepts.

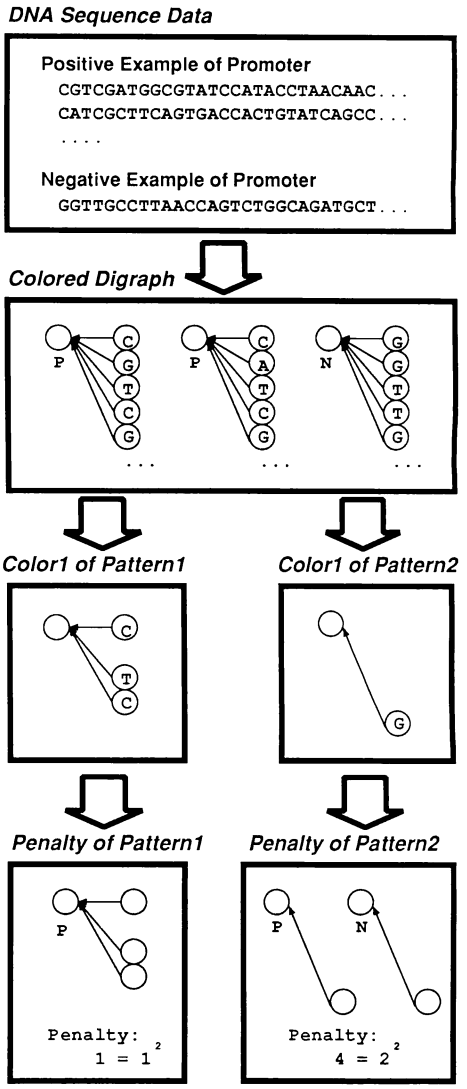


Fig. 16. Representation for mutually exclusive concepts.

Appendix B: Approximation Techniques to Generate Abstract-Level Concepts

Simpler representations can be obtained by the use of approximation techniques. Generating concepts such as digital NOT and NOR requires the use of these methods. Note that the analog NOR and NOT still have analog components, such as [s+], for the input and output values (see figure 13). However, the input and output relationships of digital NOT and NOR are exactly the same as those in the logical concepts.

We distinguish between two types of approximation in colored digraphs. Type 1 approximation involves the rule color and is equivalent to assuming some unknown function of an identified object in the circuit. In terms of graph operations, this removes the edge portion from the rule subgraph for which the inference process is of a certain type. Type 2 approximation utilizes the value color and is equivalent to assuming some default value. In terms of graph operations, this amounts to removing the edge portion from the rule subgraph for which the values of some data are the same for all occurrences.

These approximations are illustrated in figure 17 and figure 18. The second graph in figure 17 shows the state of the graph just before a type 1 approximation is introduced. Since at this stage

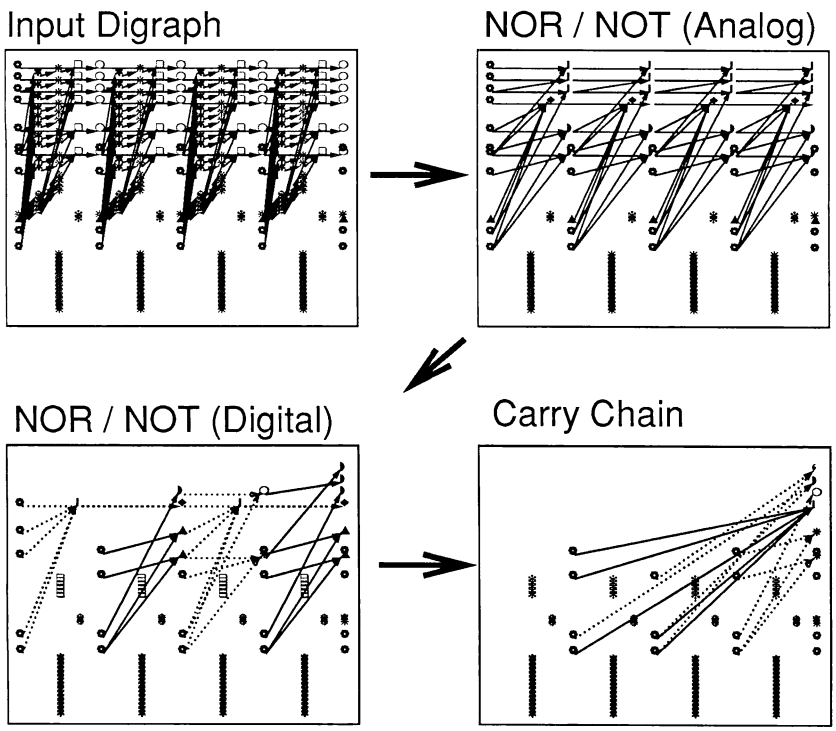


Fig. 17. Graph contraction using type I approximation.

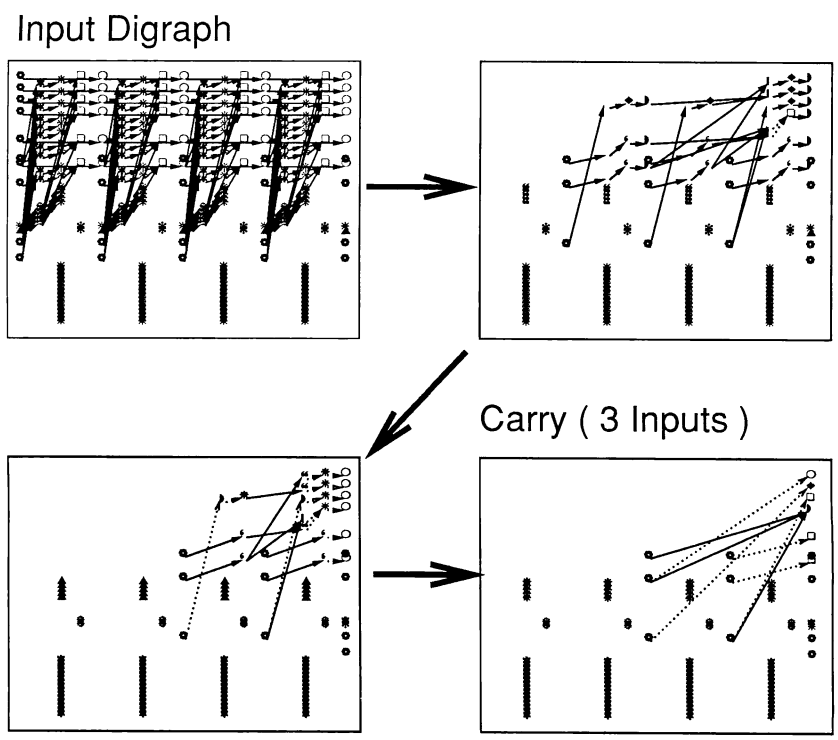


Fig. 18. Graph contraction using type II approximation.

the patterns are still small, no details are neglected, and the result is exactly the same as that shown in figure 14.

At this stage, a type 1 approximation is introduced to give the third graph. The node with single input solid line corresponds to NOT, and the node with two input solid lines corresponds to NOR. The resulting interpretation rules involve only [+] and [0]. If we interpret [+] as [true] and [0] as [false], these rules represent truth tables for NOT and NOR.

The concept of *Carry Chain* can also be found by further application of approximation methods. Consider the third graph in figure 17. The number of inputs for this Carry Chain is five; however, the actual number of inputs to a carry operation should be three. On performing a type 2 approximation by assuming a default value for the clock signal, the resulting graph clearly indicated a three-input carry chain operation (figure 18). Type 2 approximation also affected the second and third graphs in the figure, but the extracted NOT and NOR were essentially the same as before.

Notes

1. See section 6. All the results in section 6 utilized these estimations.
2. They are essentially same trace that is used in [28].
3. Note that the spatial allocations of the nodes are carefully selected by hand on the basis of readability alone. The X axis is sorted using the time step information of the simulation. The NOT circuit data are located in the upper portion of the figure. The NOR circuit data are located in the lower portion. Without this type of careful coordination, this figure is hard to understand. The algorithm, however, relies solely on topological information and does not have the benefit of the visual information.
4. To extract *Digital NOT/NOR* and *Carry Chain* concepts, approximation techniques are necessary. See appendix B for more details.

References

1. R.S. Michalski, "A theory and methodology of inductive learning," *Artif. Intell.*, vol. 20, pp. 111–161, 1983.
2. J.R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.
3. T. Ellman, "Explanation-Based Learning: A survey of programs and perspectives," *ACM Comput. Surv.*, vol. 21, no. 2, pp. 165–220, 1989.
4. S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artif. Intell.*, vol. 55, pp. 311–365, 1992.
5. B. Falkenhainer and K.D. Forbus, "Compositional modeling: finding the right model for the job," *Artif. Intell.*, vol. 51, pp. 95–143, 1991.
6. Z.Y. Liu and A.M. Farley, "Shifting ontological perspectives in reasoning about physical systems," in *AAAI-90*, Boston, MA, 1990, pp. 395–400.
7. M. Lebowitz, "Integrated learning: controlling explanation," *Cog. Sci.*, vol. 10, pp. 219–240, 1986.
8. M. Pazzani, M. Dyer, and M. Flowers," The role of prior causal theories in generalization," in *AAAI-86*, Philadelphia, PA, 1986, pp. 545–550.
9. A.P. Danyluk, "The use of explanation for similarity-based learning," in *IJCAI-87*, Milan, Italy, 1987, pp. 274–276.
10. H. Hirsh, "Combining empirical and analytical learning with version spaces," in *ML-89*, Ithaca, NY, 1989, pp. 29–33.
11. P.S. Rosenbloom and J. Aasman, "Knowledge level and inductive uses of chunking (EBL)," in *AAAI-90*, Boston, MA, 1990, pp. 821–827.
12. O. Etzioni, "STATIC: A problem-space compiler for PRODIGY," in *AAAI-91*, Anaheim, CA, 1991, pp. 533–540.
13. S.A. Vere, "Induction of relational productions in the presence of background information," in *IJCAI-77*, Cambridge, MA, 1977, pp. 349–355.
14. J.R. Anderson and P.J. Kline, "A learning system and its psychological implications," in *IJCAI-79*, Tokyo, Japan, 1979, pp. 16–21.
15. R. Levinson, "A self-organizing retrieval system for graphs," in *AAAI-84*, Austin, TX, 1984, pp. 203–206.
16. N.S. Flann and T.G. Dietterich, "A study of explanation-based methods for inductive learning," *Machine Learning*, 1989, pp. 187–226.
17. L.B. Holder, "Empirical substructure discovery," in *ML-89*, Ithaca, NY, 1989, pp. 133–136.
18. L.B. Holder, D.J. Cook, and H. Bunke, "Fuzzy substructure discovery," in *ML-92*, Aberdeen, Scotland, 1992, pp. 218–223.
19. T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli, "Explanation-based generalization: a unifying view," *Machine Learning*, vol. 1, pp. 47–80, 1986.
20. G. DeJong and R. Mooney, "Explanation-based learning: an alternative view," *Machine Learning*, vol. 1, pp. 145–176, 1986.
21. R.E. Korf, "Macro-operators: A weak method for learning," *Artif. Intell.*, vol. 25, pp. 35–77, 1985.
22. G.A. Iba, "A heuristic approach to the discovery of macro-operators," *Machine Learning*, vol. 3, pp. 285–317, 1989.
23. B. Kuipers, "Qualitative simulation," *Artif. Intell.*, vol. 29, pp. 289–338, 1986.
24. G.G. Towell, J.W. Shavlik, and M.O. Noordewier, "Refinement of approximate domain theories by

knowledge-based neural networks," in *AAAI-90,* Boston, MA, 1990, pp. 861–866.
25. B. Efron, "The jackknife, the bootstrap and other resampling plans," in *SIAM,* Bowling Green State University: Bowling Green, OH, 1982.
26. L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees,* Wadsworth & Brooks/Cole Advanced Books & Software: Belmont, CA, 1984.
27. S.M. Weiss and N. Indurkhya, "Reduced complexity rule induction," in *IJCAI-91,* Sydney, Australia, 1991, pp. 678–684.
28. S. Yamada and S. Tsuji, "Selective learning of macro-operators with perfect causality," in *IJCAI-89,* Detroit, MI, 1989, pp. 603–608.
29. C. Mead and L. Conway, *Introduction to VLSI Systems,* Addison-Wesley: Reading, MA, 1980.
30. S. Arikawa, S. Miyano, and A. Shinohara, "Knowledge acquisition from amino acid sequences by learning algorithms," in *JKAW92,* Hatoyama, Japan, 1992, pp. 109–128.
31. S. Minton, "Quantitative results concerning the utility of Explanation-Based Learning," *Artif. Intell.,* vol. 42, pp. 363–391, 1990.

Kenichi Yoshida is a research scientist at the Advanced Research Laboratory, Hitachi Ltd. He received his Ph.D. in computer science from Osaka University. He received the best paper awards from the Atomic Energy Society of Japan, from The Institute of Electrical Engineers of Japan, and from the Japanese Society for Artificial Intelligence. His current research interest includes machine learning, knowledge representation, and expert systems.

Hiroshi Motoda has been with Hitachi since 1967. He is currently a chief research scientist at the Advanced Research Laboratory, Saitama, Japan, and heads the AI group. His current research includes machine learning, knowledge acquisition, and visual reasoning. Previously he was a senior researcher at the Energy Research Laboratory, where he participated in research on core management, control and design of nuclear power reactors, and expert systems for plant diagnosis and layout planning.

He received his B.S., M.S., and Ph.D. degrees in nuclear engineering from the Univerity of Tokyo. He was on the board of trustees of the Japan Society of Software Science and Technology and the Japanese Society for Artificial Intelligence, and is on the editorial board of *Knowledge Acquisition, IEEE Expert and Artificial Intelligence in Engineering.* He is a member of AAAI, IEEE Computer Society, JSAI, JSSST, IPSJ, and JSCS. He received the best paper awards twice from the Atomic Energy Society of Japan and twice from the Japanese Society for Artificial Intelligence.

Nitin Indurkhya is an Assistant Professor of computer science at the University of Sydney, Australia. He received his Ph.D. in computer science from Rutgers University, U.S.A. His current research focuses on various issues in machine learning such as automatic model construction and time-series data analysis, with particular interest in hidden Markov modeling, rule-based induction, decision trees, neural nets, and nonparametric statistics.