CLIP: Concept Learning from Inference Patterns

Ken'ichi Yoshida [a] and Hiroshi Motoda [a]

[a] *Advanced Research Laboratory, Hitachi, Ltd. Hatoyama, Saitama, 350-03, Japan*

Abstract

A new concept-learning method called CLIP (*C*oncept *L*earning from *I*nference *P*atterns) is proposed that learns new concepts from inference patterns, not from positive/negative examples that most conventional concept learning methods use. The learned concepts enable an efficient inference on a more abstract level. We use a colored digraph to represent inference patterns. The graph representation is expressive enough and enables the quantitative analysis of the inference pattern frequency. The learning process consists of the following two steps: 1) Convert the original inference patterns to a colored digraph, and 2) Extract a set of typical patterns which appears frequently in the digraph. The basic idea is that the smaller the digraph becomes, the smaller the amount of data to be handled becomes and, accordingly, the more efficient the inference process that uses these data. Also, we can reduce the size of the graph by replacing each frequently appearing graph pattern with a single node, and each reduced node represents a new concept. Experimentally, CLIP automatically generates multilevel representations from a given physical/single-level representation of a carry chain circuit. These representations involve abstract descriptions of the circuit, such as mathematical and logical descriptions.

1 Introduction

Human beings use various abstract concepts, such as *logic* and *mathematics*, to acquire new knowledge. These concepts are crucial to achieving scientific and technical breakthroughs. We also use various concepts in daily life. For example, we sometimes complain, *He is too stubborn to negotiate.* In this case, *stubborn* represents a certain characteristic of the person, and we can deduce some conclusion, such as *Find other person to discuss the problem*, without considering the details.

How do human beings acquire these "concepts"?

Finding some answer to this question and representing it on a computer system is an important theme in artificial intelligence.

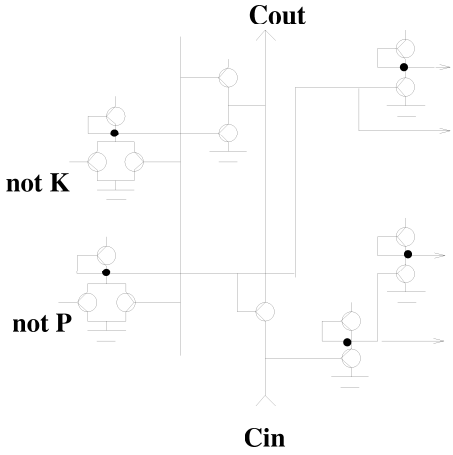
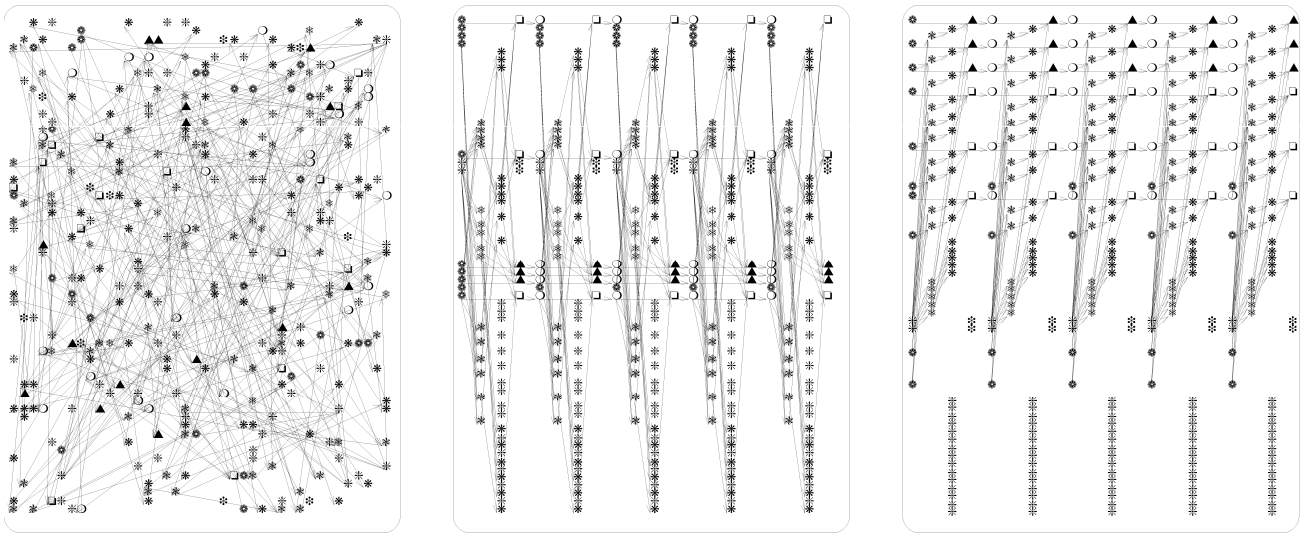


Fig. 1. Carry chain circuit



(a) Naive Inference Pattern (b) Sorted by Qualitative Reasoning Knowledge (c) Sorted by Circuit Knowledge

Fig. 2. Example of concept usage

To examine this question in more detail, we use qualitative simulation traces as an example of the concept utilization. Figure 1 shows a carry-chain circuit [9] which is part of the CPU, and Fig. 2 shows the qualitative simulation [4] results of its behavior, *i.e.*, the changes in current, voltage, etc. Note that Figs. 2(a), 2(b) and 2(c) show the same information, all displaying the dependency among the data, with the only difference among them being in the layout of the data. In Fig. 2(a), the location of each datum is randomly generated, and the arrows display how a datum is calculated from other data. In Fig. 2(b), the spatial allocation of the data is selected by hand. The X axis is sorted using the time-step information from the simulation along with mythical causality[4],

and the Y axis is sorted using the name of the data. In other words, Fig. 2(b) uses some knowledge about *qualitative simulation* to lay out the data shown in Fig. 2(a). Figure 2(c) also uses specific knowledge about the carry-chain circuit, here, the NOT circuit data are located in the upper portion of the figure, and the NOR circuit data are located below.

Which figure is best?

We think Fig. 2(c) is the best figure. It seems to be the best in various situations, *e.g.*, to explain the data dependency between the data or to memorize the whole structure. We cannot imagine a situation in which Fig. 2(a) is the best to use. Although the data that Fig. 2 shows are for the physical behavior of circuits, such as the changes in the voltage and currents, we seem to use abstract-level concepts, such as the behavior of NOR/NOT circuits, in understanding the figure. Here, we use abstract-level concepts to leave out unnecessary details by giving a new concept name to the aggregation of phenomena, and abstract-level concepts are crucial to understand this complex figure by reducing complexity.

Based on the above introspection, we assume that *A concept is something which makes inference easier*, and propose a new concept-learning method, *C*oncept *L*earning from *I*nference *P*atterns (CLIP). To find new concepts that satisfy this assumption, CLIP analyses inference processes and tries to extract typical patterns from them. Each extracted pattern corresponds to a new abstract concept by use of which inference is made efficient due to the reduced complexity. CLIP's most important characteristics is that it does not require any prespecified knowledge about abstract concepts. For example, CLIP analyses qualitative simulation traces of the physical behavior (*i.e.*, the changes in the voltage and current) of the digital circuits, and finds the logical concepts, such as NOR and NOT, without being provided any knowledge about logic. Here, the concepts NOR and NOT are generated to achieve an efficient inference.

Another important characteristic of CLIP is that it also generates a set of rules to interpret new concepts, and these interpretation rules enable abstract-level inference. This distinguishes CLIP from conventional concept learning methods, such as that in [3], which do not aim to make inference at the abstract level.

The rest of the paper is organized as follows: Section 2 outlines the basic idea of CLIP and the algorithm for extracting typical patterns, and Section 3 presents experimental results. Section 4 analyses the factors which affect the generated concepts and their structures with additional experimentation. Then, Section 5 examines related work, Section 6 discusses future issues, and Section 7 summarizes the results.

2 Concept Learning based on Colored Digraph Representation

CLIP analyzes the sample inference traces and extracts patterns in such a way that the new patterns can be used to make the inference more efficient. The learning process is outlined as follows:

- First, sample traces of the inference are converted into a colored digraph. We use a colored digraph to represent inference traces. Each graph node represents the data referred to or produced by the inference, and the direction of the graph edge represents the direction of the inference. Each node has two kinds of color. First color corresponds to the identifier of the inference rule that is used to calculate the value of the data, and second color corresponds to the value itself. In the circuit domain, first color corresponds to circuit equation number (more precisely, identifier of the interpretation rule that is used to interpret the circuit equation), and second color corresponds to the value of the voltage, current, etc. This graph representation enables the quantitative analysis of the inference pattern frequency.
- Next, based on the frequency analysis of the inference pattern, a parallel-search algorithm extracts a set of typical patterns which frequently appear in the digraph.

The basic idea is:

- We can reduce the size of the graph by replacing each frequently appearing graph pattern with a single node. Any pattern that appears often in an inference process probably represents an important concept.
- The size of the graph corresponds to the amount of the data referred to or produced by the inference engine. Accordingly, as the graph converted from the inference traces becomes smaller, the inference becomes easier.

Figure 3 summarizes this idea. The upper left part of Fig. 3 shows a simple digital circuit, and the lower part shows the inference trace that analyzes the physical behavior of the circuit. The graph edge in the lower part shows the data dependency. The numeral in each node indicates which of the interpretation rules (see Section 3.2) is used in that node to derive the value of the variable (V_i, I_i, Q_i, etc.) attached to the node. For example, the value of the voltage V_1 is calculated from the electric charge Q_1 using Rule 2 in Fig. 3, and the value of the current I_1 is calculated from voltages V_1 and V_{cc} using Rule 3. CLIP analyzes this inference trace and extracts typical patterns, such as "*The rule sequence 2, 3 and 4 is frequently used before Rule 1*", from the graph.

CLIP's output for this case is summarized in the upper right part of Fig. 3. Here, the new inference Rule 5 is used to calculate the electric charge Q_2 from Q_1, and the smaller amount of the data in the inference process decreases the cost of the inference. Furthermore, the correspondence of the new Rule 5 and

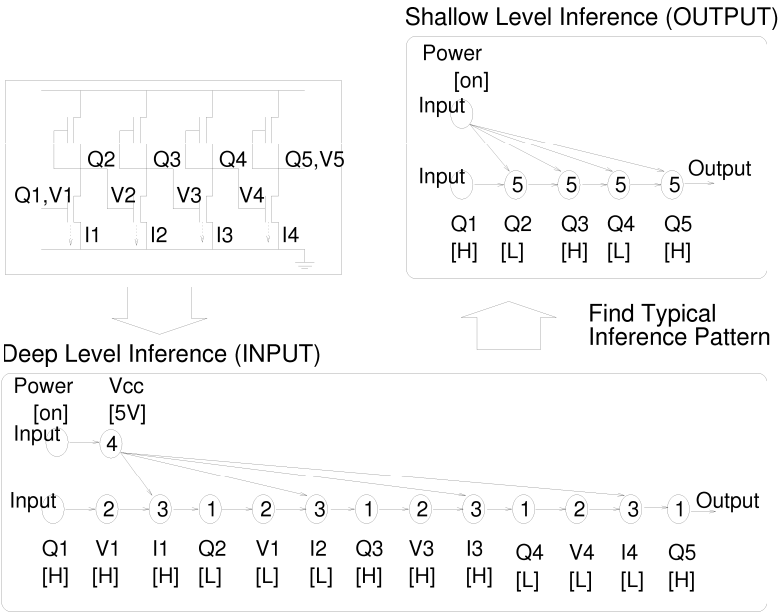


Fig. 3. CLIP: Concept learning from inference patterns

the original rule sequence 1, 2, 3 and 4 can be used to restore the original information involved in the lower graph from the upper right graph.

Note that the pattern found by CLIP corresponds to the new abstract concepts which are not explicitly represented in the original graph. In Fig. 3, the lower part of the graph represents the inference process about the physical quantities, such as the voltage and the current, and the upper right graph represents the inference process about the logical values (here, [H/L] corresponds to [True/False]). The lower part of the graph does not have any explicit information about logical behaviors.

The extraction of patterns is solely based on finding repetitions of inference patterns in a colored graph, and no semantics are considered. This makes CLIP domain-independent. [1]

2.1 Colored Digraph Representation

In the succeeding discussion, we use qualitative reasoning traces as the input inference traces. The qualitative reasoning system we used employs the entity-relationship model [2] to represent the target object. For an electric circuit, the entity is a physical datum, such as the voltage or the current at a node, and the relationship between the entities is a circuit equation that describes the

[1] See [17] for a recent study on other aspects and other application domains of CLIP.

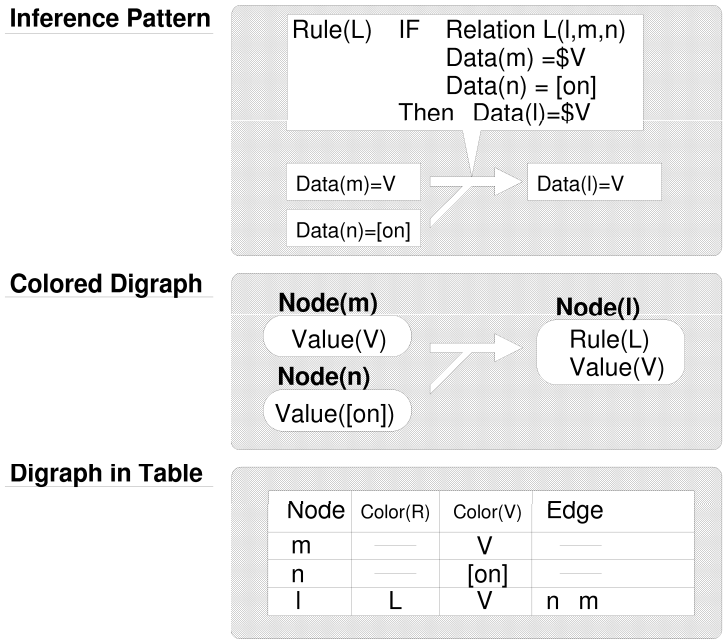


Fig. 4. Inference pattern, colored digraph and representation

relationships of the physical data. The inference is performed by interpreting the relationship, *i.e.*, the circuit equation, and calculating new values of the physical data. We also use a set of interpretation rules to describe how to use the circuit equations in calculating a new value for each data. These rules describe the inference method for qualitative simulation.

Figure. 4 shows an inference pattern, the corresponding colored digraph representation, and the table representation used to implement the CLIP program. In Fig. 4, the value of *Data(l)*, *i.e.*, V, is calculated from the value of *Data(m)* and *Data(n)* using *Rule(L)*. *Data(m)* corresponds to *Node(m)*, and the color of the node, *Color(R)*, identifies the interpretation rule used to calculate the data value. Here, the incoming edges are ordered, and this edge position is also stored. (See Section 2.2.1 and 3.3 for how to use the information about the edge position.)

Figure 5 shows partial results of qualitative simulation for a carry-chain circuit. The left-side graph shows the interpretation rule used to calculate the value of the data and the data dependency between the data. The right-side graph shows the qualitative values of the data.

Similarly to Fig. 2 (c), X axis is sorted using the time-step information from the simulation along with mythical causality, and the Y axis is sorted using the name of the data. The left most names, *e.g.*, v-op, i-c-oppd and etc., are the name of the data. The color of the graph node ($*$, $\square$, etc. in Fig. 5) in the left-side indicates the interpretation rule used to calculate the value of the data, and the arrows show the dependency between the data. The qualitative

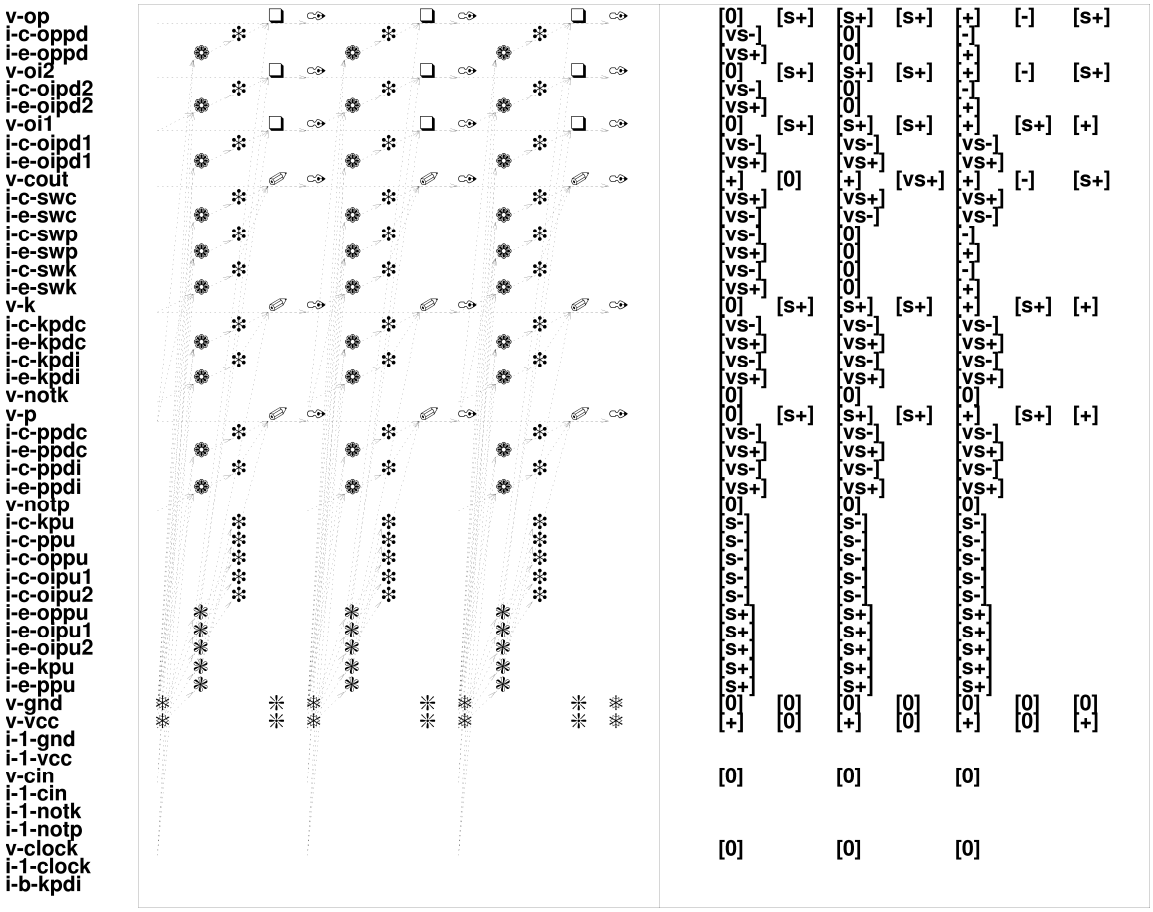


Fig. 5. Sample input digraph

values shown in the right-side are: [+], [0], [-], [s+](small +), [vs+](very small +), [s-], and [vs-].

This graph contains the information about data dependency, as well as information about how the interpretation rules calculate the output values from inputs. For example, the upper-most part of Fig. 5 shows that : the initial value of the voltage v-op is [0] (first line, first value), and the current i-c-oppd (the second line of the figure), i-e-oppu (the 34th line) and the interpretation rule □are used to calculate the next value (⌊s+⌋: first line, second value) of v-op.

In Fig. 5, the interpretation rule □(qualitative inference rule about the additive relationship) is used nine times. By analyzing the values of the variables in each of these occurrences, we can extract a qualitative inference rule about the additive relationship. In this case, the extracted rule is a subset of the additive relationship, *i.e.*, "⌊vs-⌋+⌊s+⌋→⌊s+⌋", "⌊0⌋+⌊s+⌋→⌊s+⌋", and "⌊-⌋+⌊s+⌋→⌊-⌋", and the missing relationship such as "⌊-⌋+⌊0⌋→⌊-⌋" is not necessary for the inference that corresponds to the input graph.

The same technique can be applied to extract larger patterns, and this is what CLIP basically does. We can use the graph shown in Fig. 5 as the input graph. Since different traces are obtained for different initial values for the same circuit, there can be more than one trace. CLIP can take as many graphs as necessary. Hereafter, we call these graphs collectively an input graph, *i.e.*, the input graph can be a single large graph or a collection of small graphs.

The CLIP user may select a subset of a graph to be processed by CLIP. It may help to speed up learning by decreasing the amount of data. However, such preprocessing is not crucial to CLIP because it learns new concepts from a graph that represent certain characteristics of the environment, and thus, examples are embedded in the graph.

Note that the spatial allocations of the nodes are carefully selected by hand on the basis of readability alone in Fig. 5. CLIP uses only topological information and does not have the benefit of visual information. In other words, CLIP only uses information involved in Fig. 2(a), and does not use domain-specific knowledge such as knowledge about the carry-chain circuit.

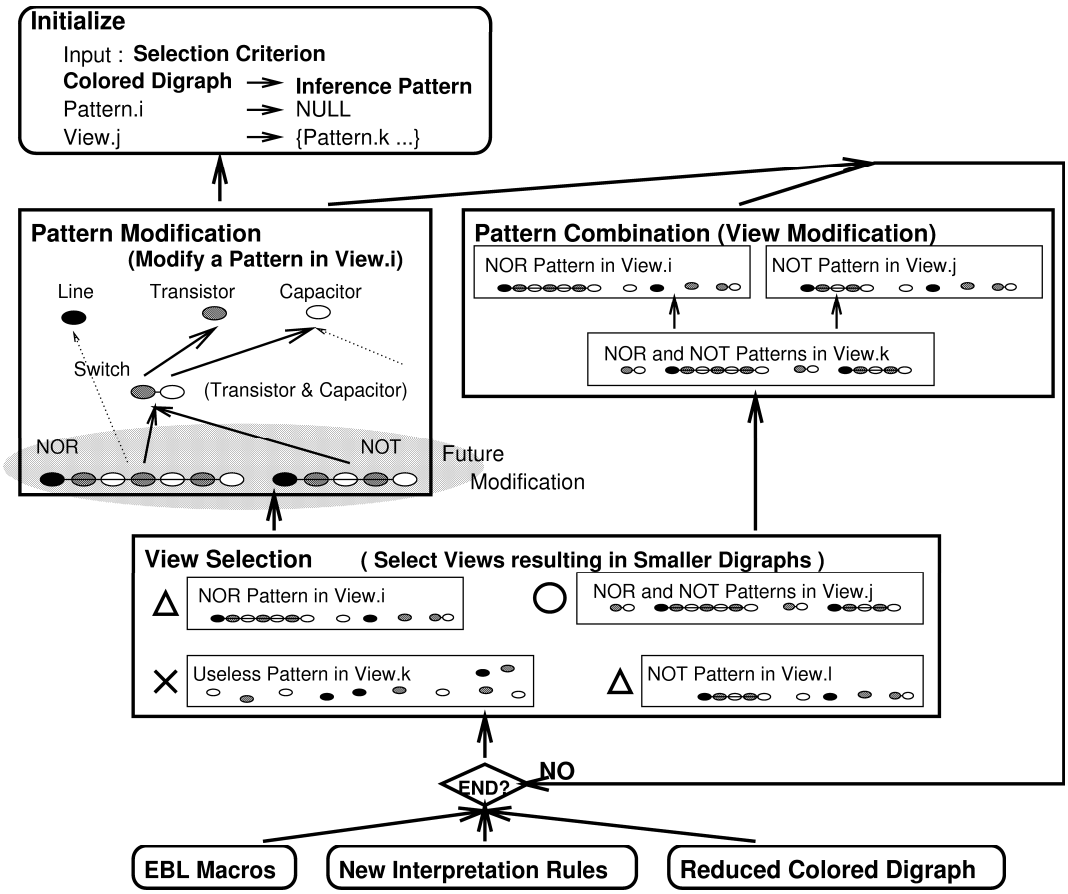
2.2 Algorithm for Finding Typical Patterns

After the learning problem is encoded as a colored digraph, the key step is to extract typical patterns. (See Fig. 6.) Each pattern is interpreted in two ways: as an EBL macro rule, and as an interpretation rule. (See Section 3.2 for details.) In this section, we describe in detail the algorithm, called CLIP, for performing this task. CLIP is a beam-search algorithm, amenable to parallel implementation, that searches for typical patterns in the colored digraph. The objective is to find typical patterns that help rewrite the graph. In order to assist in the choice of desirable typical patterns, a selection criterion for comparing alternative rewritten graphs is provided to the CLIP algorithm. The search procedure focuses on obtaining a reasonably good solution, not necessarily the optimal one.

Two different data structures, *View* and *Pattern*, are used to store the set of candidate patterns. Here, a *Pattern* stores a single pattern, and a *View* stores a set of patterns. [2] In the digital circuit domain, *Pattern* stores a graph pattern that corresponds to a certain circuit behavior, *e.g.*, behavior of NOR and NOT. *View* stores a set of these *Pattern*s.

The method involves three basic operations: *Pattern Modification*, *Pattern*

[2] The current program uses array structure to implement *View*, and a special *Pattern*, called null pattern, is used to indicates the absence of the pattern in the *View*.



The mark in View Selection indicates the result of graph rewriting: ◯ for good, × for bad and △ for intermediate.

Fig. 6. Algorithm

Combination and *View Selection*. It starts with N views, and iteratively extends the patterns in the view by the first two operations (the old patterns are also retained). Here, N is an input parameter to specify the beam search width. In each iteration, the input graph is rewritten using patterns in each view and only the good views are retained. The heart of the CLIP procedure involves iteratively performing these basic operations.

2.2.1 Pattern Modification

In *Pattern Modification*, first a view is selected and the digraph is rewritten according to the patterns in the selected view. Each occurrence of the pattern in the input graph is replaced by a single node (Figs. 7(a) and 7(b)). This graph-rewriting procedure reduces the size of the graph by replacing each complex graph pattern with a single node. Accordingly, as the graph converted from the inference traces becomes smaller, the inference becomes easier.

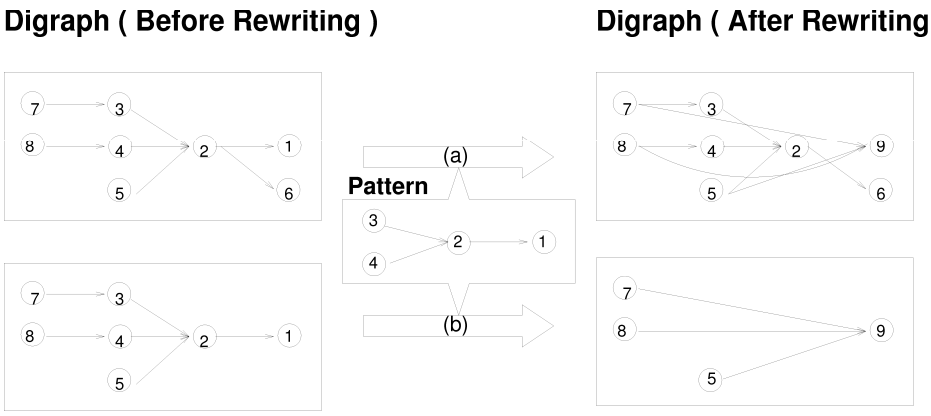


Fig. 7. Digraph rewriting by pattern substitution

Note that this graph-rewriting operation is different from the graph contraction in the following three ways:

(i) 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 incoming edges are ordered, and only the equivalence between the corresponding edges is examined. An important implication of this is that graph-matching can be done in polynomial time. 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). This restriction does not seem to limit the class of learning tasks that the algorithm can handle.
(ii) The graph-matching procedure also checks the equivalence of node color. This also makes the matching efficient by further pruning the candidates.
(iii) If the data from the intermediate graph node is used to calculate another value (Fig. 7(a)), the original node is retained.

Figure 8 shows how patterns are modified in this step. The reduced graph is analyzed and every possible pattern made up of two linked nodes is considered. These patterns are referred to as temporary patterns. Each such temporary pattern is then expanded based on patterns in the current view, and used to create new views. In the example, 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. By rewriting an expanded pattern first and then rewriting the original pattern, we get a smaller graph even if the expanded pattern does not rewrite all the occurrence of the original pattern.

In the circuit domain, node color corresponds to circuit equation number (more precisely, interpretation rule number). The new color, therefore, corresponds to a new circuit equation which is translated from original equations. CLIP

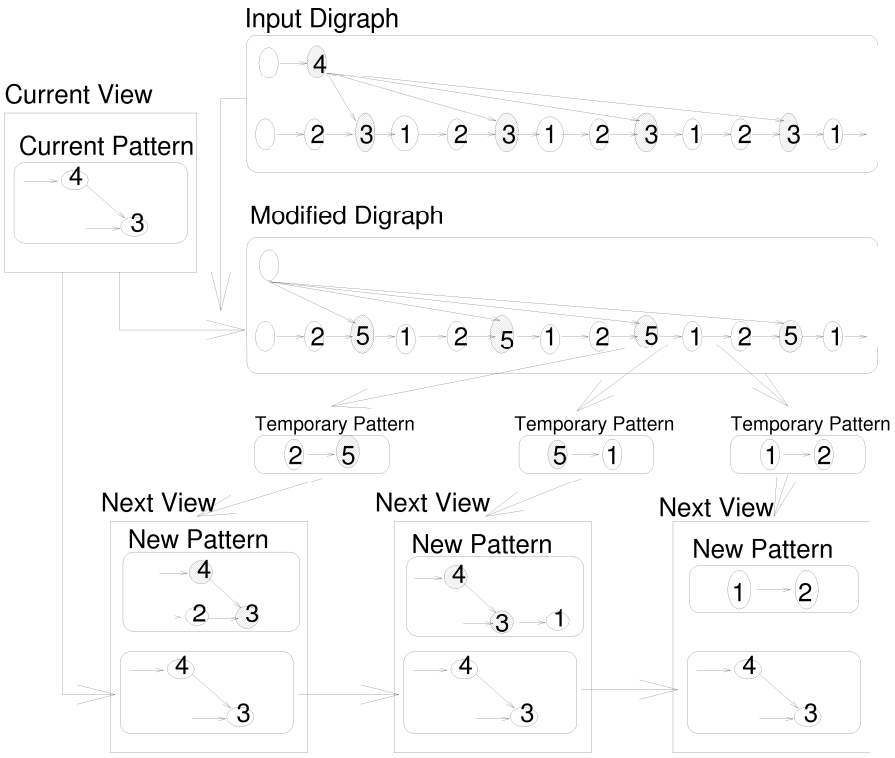


Fig. 8. Pattern modification

also uses node color which corresponds to the value of the data. However, pattern modification does not use this color (See Section 3.2 for this color).

If the capacity of the view is exceeded, CLIP discards low-priority patterns judging from the result of *(frequency of usage)* × *(size of pattern)*. Note that *Pattern modification* modifies *Pattern* in a stepwise manner. A single invocation of this operation modifies only one pattern, and therefore, multiple invocations are required to generate a complex pattern, *e.g.*, a pattern that corresponds to NOR/ NOT circuits.

2.2.2 Pattern Combination

Pairs of existing views are combined to obtain new views. All possible combinations are considered. Again, if the capacity of the view is exceeded, low-priority patterns are discarded.

2.2.3 View Selection

Estimates are obtained for each view as to how much reduction in graph size can be expected after rewriting the input graph using patterns in that view, and only highly ranked views are chosen, up to the allowable number of views.

Note that the definition of the graph size is an important input for CLIP. By changing this definition, CLIP can find various concepts from the same inputs. (See Section 4 for the details.)

Figure 9 illustrates how patterns evolve through iterations. In this example, the maximum number of views is limited to 4. In the first iteration, 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 iterations, in addition to pattern modification, pattern combination 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 views (maximum number of views, which is a search parameter; 4 in Fig. 9) in each iteration.

View selection is based on estimates of the relative effectiveness of views in graph-rewriting. These estimates are computed as follows: For each of the views $V_{Previous}$ (that were selected in the previous iteration), the actual size $C_{Exact}(V_{Previous})$ of the rewritten graph that results from applying patterns in that view to the graph is calculated at the beginning of the current iteration. For each new view $V_{Current}$ that is generated in the pattern-modification step, the estimated size $C_{Estimate}(V_{Current})$ of the rewritten graph that would result from applying that view is calculated as a perturbation to the actual size of the rewritten graph due to the parent view $V_{Previous}$. The graph size for views generated in the pattern-combination step is estimated as a linear combination of the two views involved in the combination.

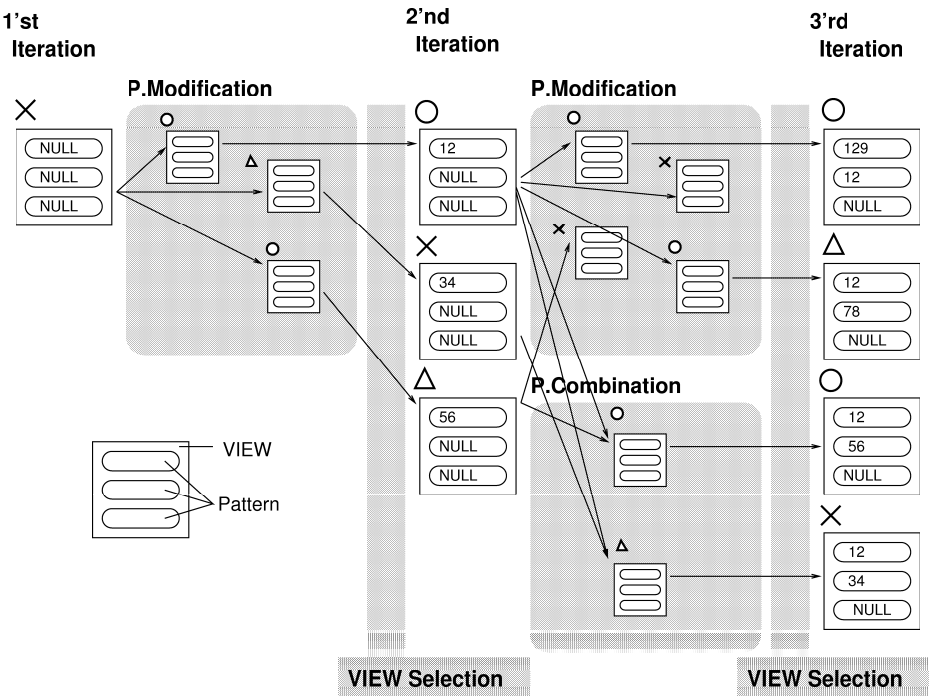
The estimated size and the actual size (calculated at the beginning of the next iteration) may not agree. (See the difference between big and small marks at the top left of each view in Fig. 9.) However, the estimate is reasonable enough to ensure that good views are usually selected. The estimation process is necessary because computation of the actual size is computationally expensive. The graph size is estimated in pattern modification by

$$C_{Estimate}(V_{Current}) = (1.0 - \alpha \times F) \times C_{Exact}(V_{Previous})$$

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

In the case of the pattern combination,

$$C_{Estimate}(V_{Current}) = (1 - \beta) \times C_{Exact}(V_{Previous1}) + \beta \times C_{Exact}(V_{Previous2}),$$



The mark at the upper left of each view indicates the result of graph rewriting: ◯ for good, × for bad and △ for intermediate. A large mark indicates a result based on actual rewriting of the graph; a small one, on estimation.

Fig. 9. Data flow

where α and β are input parameters for the experiments. (See Sections 3 and 4 for the values and the results.) Note that graph-rewriting is performed only at the pattern modification step (see Fig. 8), and is limited to the maximum number of views. If the graphs were actually rewritten during view selection, more than 500 rewriting would be required in circuit domain examples, while the maximum number of views was set at 15 (see Section 3.2). Obviously, then this size prediction represents a great cost savings.

The pattern-finding algorithm explained in this section can be seen as a kind of Genetic Algorithm[7]. Here, *View* corresponds to *Chromosome*; *Pattern*, to *Gene*; *Pattern Modification*, to *Mutation*; *Pattern Combination*, to *Crossover*; *graph size*, to *fitness function*; and *View Selection*, to *Selection*.

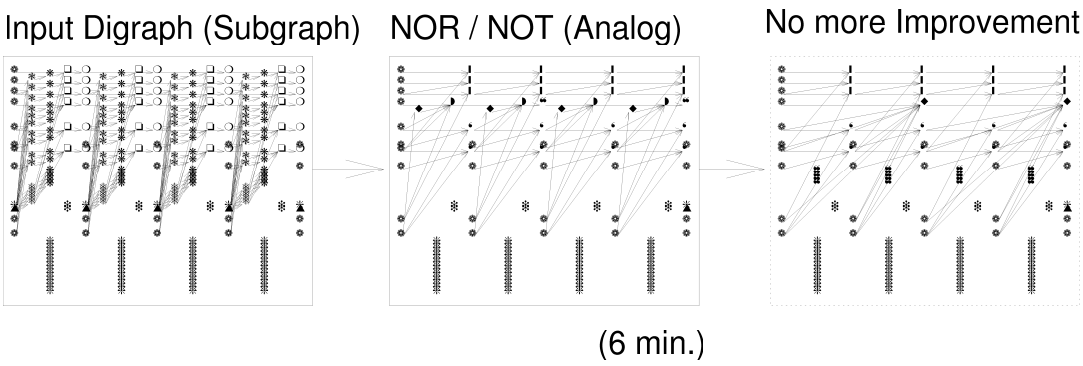


Fig. 10. Experimental result (resulting digraph)

3 Outline of Experimental Results

3.1 Input and Parameters

We implemented the pattern-finding algorithm CLIP, described in the previous section, using C programming language. An example of the graph-rewriting process performed by CLIP is shown in Fig. 10. The input graph was converted from the results of a qualitative simulation of the behavior of the carry-chain circuit shown in Fig. 1. The total number of nodes in the input graph was 2176 and the total number of edges was 2144. The carry-chain circuit has three boolean inputs, and the input graph involves the simulation results of eight ($= 2^3$) cases. The left-most part of Fig. 10 shows a subset of the input graph that corresponds to one such input case.

The qualitative values used in the qualitative simulation are: [+], [0], [-], [s+](small +), [vs+](very small +), [s-], and [vs-]. By using this information about the magnitude of the value, the qualitative simulation system can express information such as: *The current through the pullup transistor is smaller than the current through the pulldown transistor.*

The graph size was evaluated using the following equations:

$$
\begin{aligned}
&C_{Exact} = \\
&\text{Number of Nodes} \\
&+\text{Number of Edges} \\
&+\sum (\text{Number of I/O Combinations of the Interpretation Rule}_i)^2 \\
&+\sum (\text{Number of Inputs to the Pattern}_i)^2 \\
&-\text{Number of Graph-Rewritings} \qquad (1)
\end{aligned}
$$

$C_{Estimate}$ for *Pattern Modification* =

$$(1.0 - 0.1 \times F) \times C_{Exact}(V_{Previous})$$

$$C_{Estimate} \text{ for } \textit{Pattern Combination} =$$
$$(1 - 0.5) \times C_{Exact}(V_{Previous1}) + 0.5 \times C_{Exact}(V_{Previous2})$$

With this definition, the smaller the rewritten digraph becomes, the less the inference cost becomes. The first term in equation (1) represents the amount of data to be handled; the second term represents the matching cost during the inference. The third term represents the amount of memory required to store the interpretation rules. For example, the numbers of I/O combinations of the interpretation rule of NOT and NOR are 2 ("¬False→True" and "¬True→False") and 4 ("False∨False→True", "True∨False→False", "False∨True→False" and "True∨True→False"), respectively. If the view is made up of two patterns each corresponding to NOR and NOT, the third term becomes 20 ($= 2^2 + 4^2$). As with the first term, an increase in this term results in increased memory usage. Increasing the amount of input data increases the matching cost to use the interpretation rules, and the fourth term represents that matching cost. For example, the NOT receives 1 datum, and NOR receives 2 data, so the fourth term becomes 5. [3]

The last term is to accelerate the pattern search. In the early iterations of the search, the patterns found are not big enough to sufficiently reduce the graph size, and increases in the third and fourth terms sometimes hinder the search process. The purpose of this term is to enforce the succeeding search that follows the frequently occurring patterns used to rewrite the input graph. In the later iterations of the search, the effect of this term can be neglected because a pattern which results in great reduction of the size tends to result in small values for the first and the second terms.

3.2 Results

We set the maximum number of patterns per view at 7 and the maximum number of views per iteration at 15. In an experiment with 50 iterations, the minimum size was reached at the 25th iteration. The right-most part of Fig. 10 shows the graph generated in a later iteration. The pattern found is bigger than that of 25th iteration. Increases in the third and fourth terms prevented further improvement.

[3] Note that both the number of nodes and edges were about 2000, and the number of I/O combinations and inputs of the pattern were $1 \sim 10$. To compensate for the difference of the order of magnitude of the term value, the third and forth terms are squared.

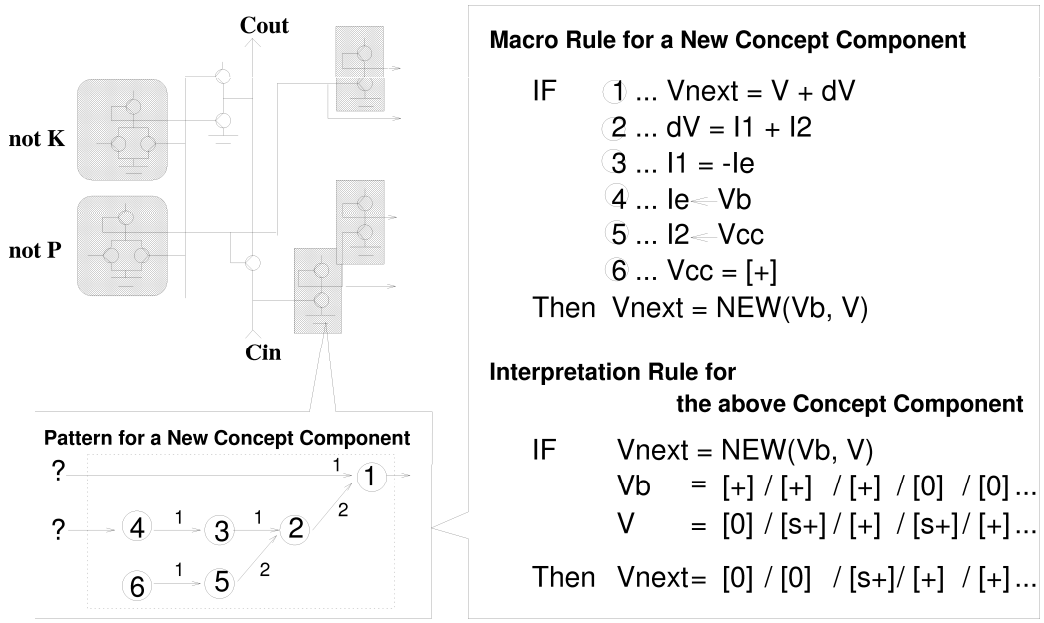


Fig. 11. Resulting patterns

In Fig. 11, we show an example of a typical pattern extracted (lower left) and the corresponding macro/interpretation rules (right) at the 25th iteration. Computation time for this 6-node pattern was about 6 minutes on a 10-mips computer. The corresponding equations used during the simulation relate to 1) voltage changes at the terminal, 2) voltage changes at the terminal caused by the incoming current, 3) collector and emitter currents in the transistor, 4) emitter current and base voltage, 5) collector current and V_{CC}, and 6) knowledge about V_{CC}. Equations 3 and 4 are for a pulldown transistor and equations 5 and 6 are for a pullup transistor. The variables on the right-hand side of each equation in the conditional portion of the extracted concept (*e.g.*, V and dV in equation ①) are arranged so that they correspond to the numbers indicated on the edges pointing to each node (*e.g.*, edges 1 and 2 going into the node ①).

The macro rule says that if there is a set of 6 relations (equations) as shown, it is reasonable to infer that V_{next} can be calculated by some relation NEW from the current values of V and V_b. The interpretation rule shown describes a set of relationships between the inputs V and V_b, and the output V_{next}. In other words, this interpretation rule defines the new function NEW. The I/O combinations stored in the interpretation rule are extracted from the occurrence of the pattern in the graph. Since this is assumed to be a NOT circuit and [+]/[0] should correspond to [True]/[False], an analog component, such as [s+] (an intermediate value between [+] and [0]), makes this representation imperfect. However, this is still quite similar to NOT, and the interpretation rule specifies the method of inference using this pseudo-NOT. (See Section 4.2 for another "NOT" which does not have this defect.)

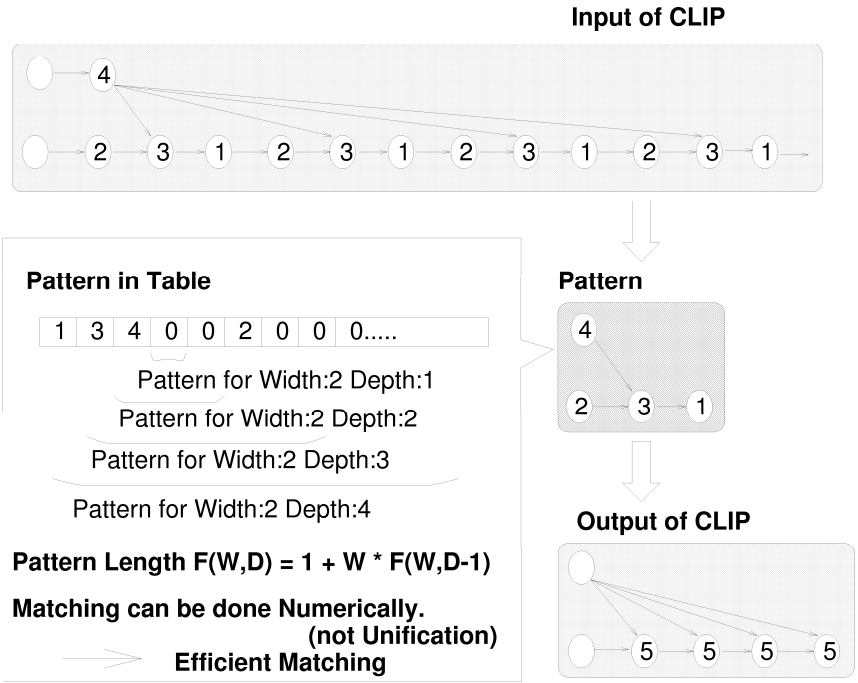


Fig. 12. Pattern representation

The macro-rules generated by CLIP can be used to translate the physical-level circuit descriptions into higher-level descriptions. For example, the macro-rule in Fig. 11 translates the circuit equations into pseudo-NOT descriptions. In another result from the same experiment, the macro-rule corresponding to NOR descriptions was also produced. In the examples shown in Figs. 10 and 11, the descriptions generated by the macro-rules and the interpretation rules enable the logical inference. If the purpose of the simulation is to predict the logical circuit behavior, this higher-level description suffices for this purpose, and is more efficient as well. Note that the proposed method does not utilize any supplementary information that explicitly defines the hierarchical structure in order to produce multilevel descriptions. It only tries to minimize the graph size.

3.3 Analysis of CPU time and Memory Requirement

CLIP can be seen as a kind of EBL system which uses efficient graph-matching as a substitute for unification. (See [12] for the relationship between unification and EBL.)

Fig. 12 shows the data structure used to implement *Pattern* in C programming language. It shows the memory requirement, *F(W,D)*, for a single pattern with

this implementation:

$$\begin{aligned} F(W,D) &= 1 + W * F(W, D-1) && \text{if D>1} \\ &= 1 && \text{if D=1} \end{aligned}$$

Here, D is the maximum depth of the pattern, and W is the maximum width of the pattern. We used 6 for D and 4 for W for the search shown in Fig. 10. The color 0 matches any color in this implementation.

If we note the correspondence between CLIP and EBL, the node color represented by a number corresponds to a rule, the edge position corresponds to a variable name, and the color 0 means any rule. Although the EBL system uses unification or an equivalent operation for the pattern-matching to learn from an example, CLIP compares numbers to extract patterns from the input graph. This was also the motivation behind using graph identity as the matching criterion, which has made the matching quite efficient.

In the experiments, the most important factor affecting the CPU time requirement was the size of the input graph. CPU time is roughly proportional to the graph size. However, the effect of pattern size, *i.e.*, the number of nodes in the pattern, on the CPU time requirement was not clear from our experiments. In Fig. 13, showing one set of experimental results, the sizes of the pattern and view increase in proportion to the iteration number. Although the CPU time needed to rewrite a single occurrence of a single pattern increases in proportion to the size of the pattern, the decrease in the pattern occurrence rate compensates for the increase in the total CPU time for rewriting the input graph.

Note that the direct implementation of the pattern shown in Fig. 12 requires a large memory when D and W are large. However, the values are almost always 0, and we can use a hash table that uses *F(W,D)* as the key to reducing the memory requirement.

4 Factors for Generating Concepts

The basic idea of CLIP is simple: to extract a set of typical patterns which appear frequently in the known inference process. In this section, we analyze various factors which may affect the generation of new concepts.

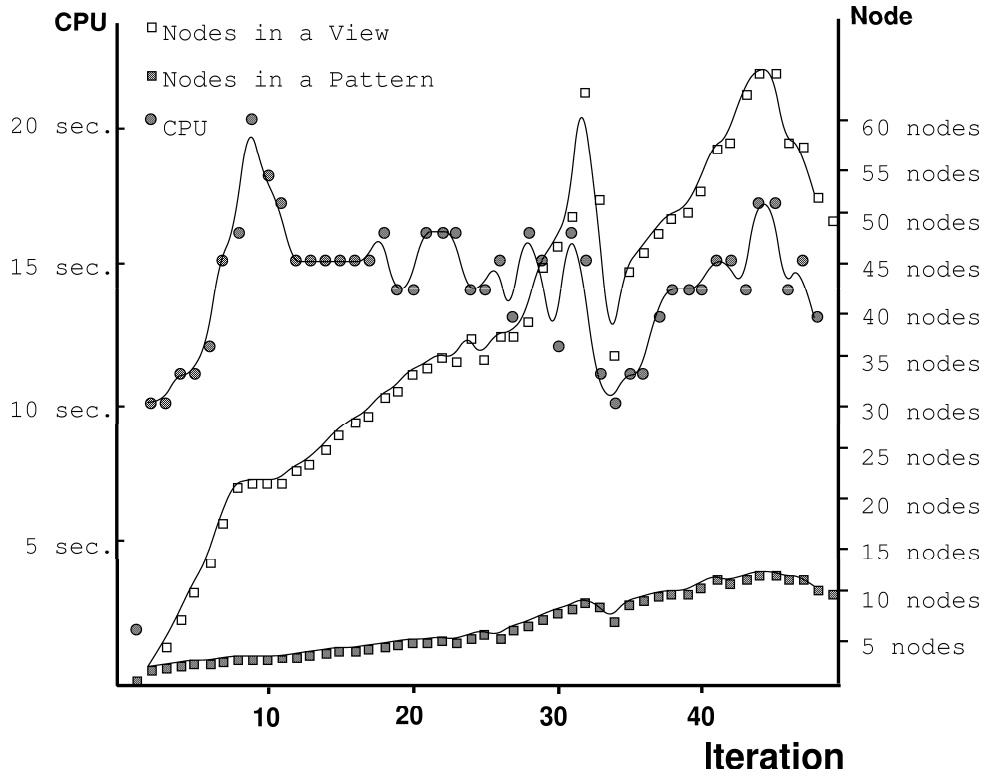


Fig. 13. Relationship between pattern length and CPU time

4.1 Environments

As described in Section 2, CLIP simply extracts patterns which originally occur in the input graph. The function of the algorithm described in Section 2.2 is not generation, but extraction. However, we believe that many existing concepts, *e.g.*, concepts used in physics, are patterns that exist in the environment of human life. Humans are able to extract a pattern from their environments as a concept to explain a certain aspect of the environment. We believe that humans' behavior is similar to CLIP's in this sense.

CLIP is designed to generate concepts that can speed up inference in some specific environment. [4] It has two important inputs: colored digraph and selection criterion. The first input, colored digraph, contains the information about the pattern of inference and their frequencies, thus it represents some characteristics of the inference system's environment.

[4] Clearly, CLIP does not cover all human abilities to create new concepts. For example, humans can compare the importance of individual data, and delete the noise from the data. They can also extend their reasoning to problems outside the current environment. Although CLIP lacks these human abilities, we believe that it covers an important aspect of concept generation.

The second input, *i.e*, selection criterion, controls the behavior of CLIP by supplying the method to evaluate extracted patterns. Therefore, the design of Equation (1), *i.e.*, selection criterion (See Section 3.1), is important to control CLIP's behavior. Its first and third terms represent the amount of data to be handled, and the second and fourth terms represent the matching cost during the inference under a given environment. If the characteristics of the environment do not change, the resulting concepts reduce some of the inference costs.

However, Equation (1) does not cover all the factors which affect the performance of the inference. For example, to perform an inference with found concepts, the original descriptions should be reexpressed using the found concepts. The parameter also varies with the implementation of the inference system. The rest of this Section discusses some of the remaining factors, using the results of additional experiments.

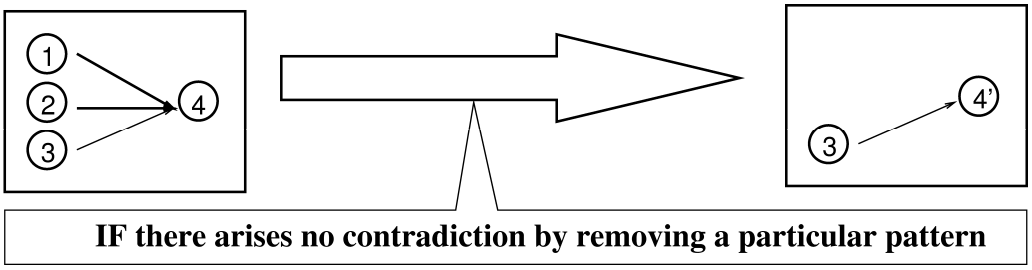
4.2 Approximation

We have already shown experimentally (Fig. 11) that the obtained interpretation rule does not exactly match a logical NOT function. This is because the magnitude of each variable is taken into account in the qualitative simulation. Even if a NOT circuit receives [+] as an input, it cannot change the output directly to [0], and it must go through [s+].

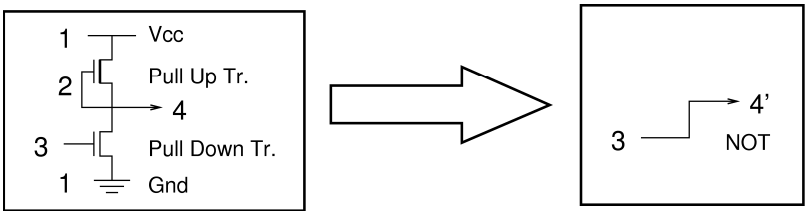
For the carry-chain circuit to work as a logical circuit, a small amount of time must elapse. Humans seem to recognize the object behavior by neglecting such intermediate state changes, and a simpler representation can be obtained by ignoring the unnecessary details. Therefore, we need some kind of approximation to go from the analog world to the digital world.

We have considered two such approximations, both involving a kind of pattern reduction (See Fig. 14). Type I approximation removes subpattern from the found pattern if the output value of the found pattern can be calculated without the information about variables in the subpattern. Type II approximation removes the edge portion from the rule subgraph for which the values of some data are the same for all occurrences. These approximations remove a pattern from the original graph, as long as no contradictions arise.

The second graph in Fig. 15 shows the results when a Type I approximation is introduced. These results are exactly the same as those shown in Fig. 10. Using this as the initial pattern, CLIP obtained the third graph using a Type I approximation. Nodes with a single-input solid line correspond to NOT and nodes with two-input solid lines correspond to NOR. The resulting interpretation rules involve only [+] and [0]. If we take [+] as [True] and [0] as [False],

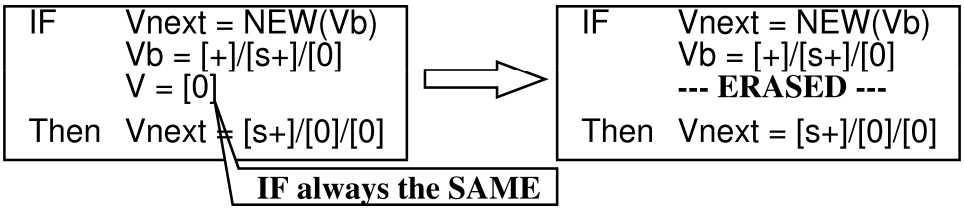


TYPE I : Utilize Rule Color (Equation Number)



Equations 1, 2, 3, and 4 form a new equation (NOT circuit).
However, the variables in eq. 1 and 2 are not necessary to calculate the output.
In such case, CLIP ignores unnecessary equations,
and generates a simpler representation.

TYPE II : Utilize Value Color



In the carry chain circuit, the clock signal is alway [0].
In such case, CLIP ignores the value of the signal,
and generates a simpler representation.

Fig. 14. Approximation

they correspond to the truth table for NOT and NOR.

The fourth graph in Fig. 15 was obtained starting from the third graph, and corresponds to a five-input carry-chain operation. However, the actual number of inputs to a carry-chain circuit should be three. When we added a Type II approximation by assuming a default value for the clock signal (Fig. 16), the resulting graph clearly indicated a three-input carry-chain operation, as well as the digital NOT and NOR. The Type II approximation also affected the second and third figures, but the extracted NOT and NOR were essentially the same. The only difference was that some parts of the graph were neglected,

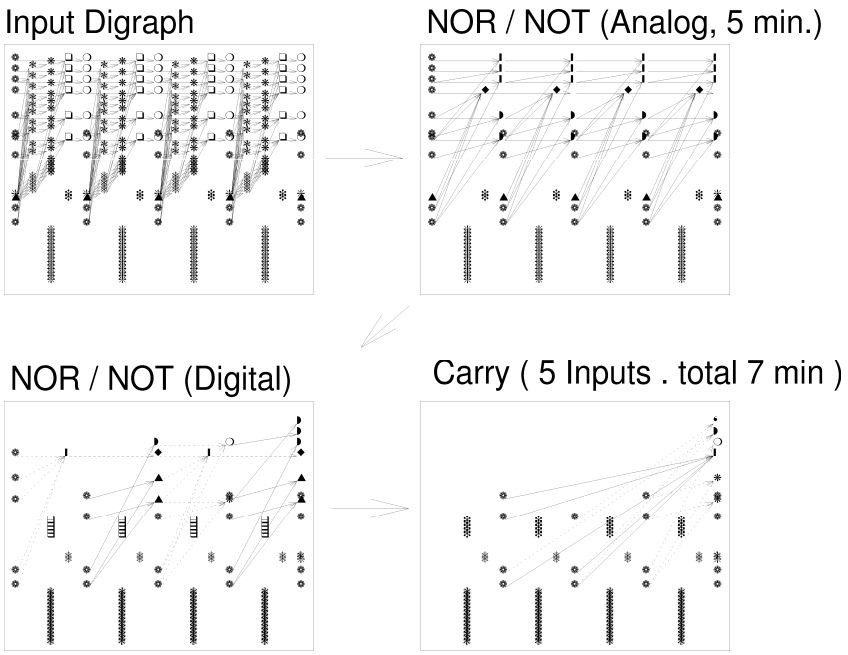


Fig. 15. Importance of approximation (TYPE I)

since the value of the corresponding physical node did not change during the simulation.

Note that the characteristics of the environment enable these approximations. If an important characteristic of the environment changes, these approximations result in contradictions. For example, if the inference system is required to treat an intermediate state change of the analog circuits, the logical concepts NOR and NOT do not help the analysis.

4.3 Inference System Characteristics

Another important factor affecting new concept generation is the inference system characteristics. To confirm the effect of the inference system characteristics on the generated concepts, different graph size definitions were used in other experiments:

$C_{Exact} =$
Number of Nodes
+Number of Edges

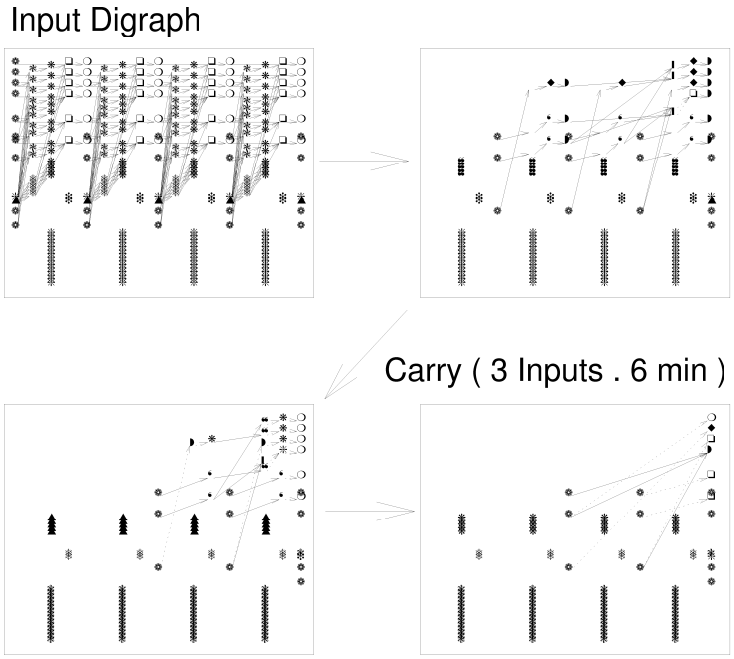


Fig. 16. Importance of approximation (TYPE II)

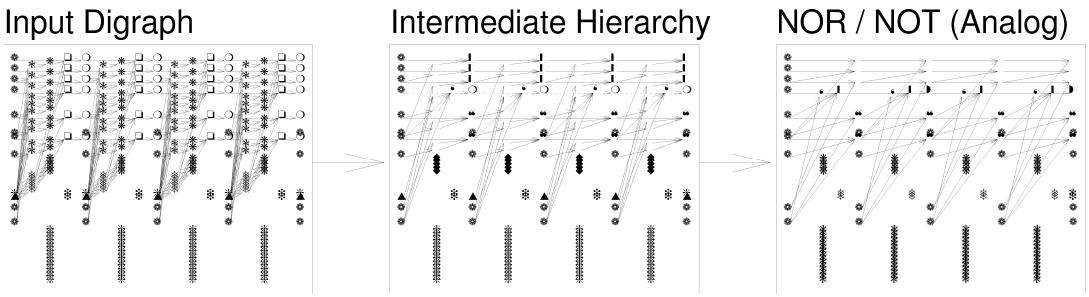


Fig. 17. Importance of rewriting costs

$$+\sum (\text{Number of I/O Combinations of the Interpretation Rule}_i)^2$$
$$+\sum (\text{Number of Inputs to the Pattern}_i)^2$$
$$+10 * \sum \text{Number of Nodes in Pattern}^2$$
$$-\text{Number of Graph Rewritings} \qquad (2)$$

The fifth term is a penalty for reexpressing the original descriptions using the found concepts. To perform an inference with the found concepts, the original descriptions should be reexpressed. As the number of nodes in the pattern

increases, the cost of reexpressing, *i.e.*, the graph-rewriting cost, increases proportionally. Although Equation (1) ignores this reexpressing cost, Equation (2) has a large penalty for it.

With Equation (2), the minimum size was reached at the 15th iteration, which is shown as the middle graph in Fig. 17. Each pattern is smaller than that of the previous result (Fig. 10), and corresponds to an intermediate physical structure, *i.e.*, a pullup or pulldown transistor. Starting from this pattern, CLIP obtained the rightmost graph, which is exactly the same as in the previous result (Fig. 10). This suggests that the optimal concept structure varies with the inference system characteristics. The more efficient the rewriting capability of the inference system becomes, the more efficient the inference becomes for fewer levels of hierarchy. CLIP can accommodate each inference system characteristic by adjusting the parameters in the size evaluation.

4.4 Color of the Graph Node

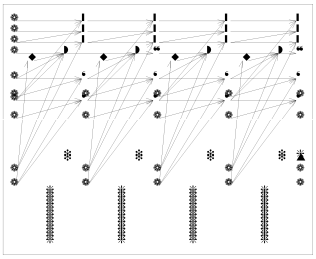
In the previous experiments, *Pattern Modification* used the rule number to find patterns. In this section, we investigate the possibility of using other information to generate concepts.

Figure 18(a) shows partial results for the case using the rule number coded in the node color, and Fig. 18(b) shows partial results for the data attribute (*i.e.*, voltage, current, etc.). Finally, Fig. 18(c) shows complete results for the data value (*i.e.*, [+], [0], etc.). As described in the previous sections, the results with the rule number involves the logical concepts NOT and NOR (Fig. 18(a)).

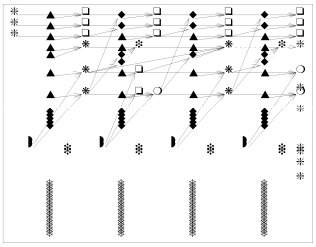
CLIP extracts some patterns even when the data attribute is used as the node color. One of them can be interpreted as *"The voltage at the input node of a transistor affects the current at the output of the transistor"*, and it represents some knowledge about transistor behavior. The relationship between the data attribute and the corresponding variables in the circuit equation seems to be a cause of this pattern generation.

When we use the value of the physical data, the resulting graph also represents some patterns (Fig. 18(c)). In this case, each pattern represents the distribution of the value in the circuit. For example, the two right-most graphs in Fig. 18(c) indicate the similarity of the calculations performed in corresponding simulation cases.

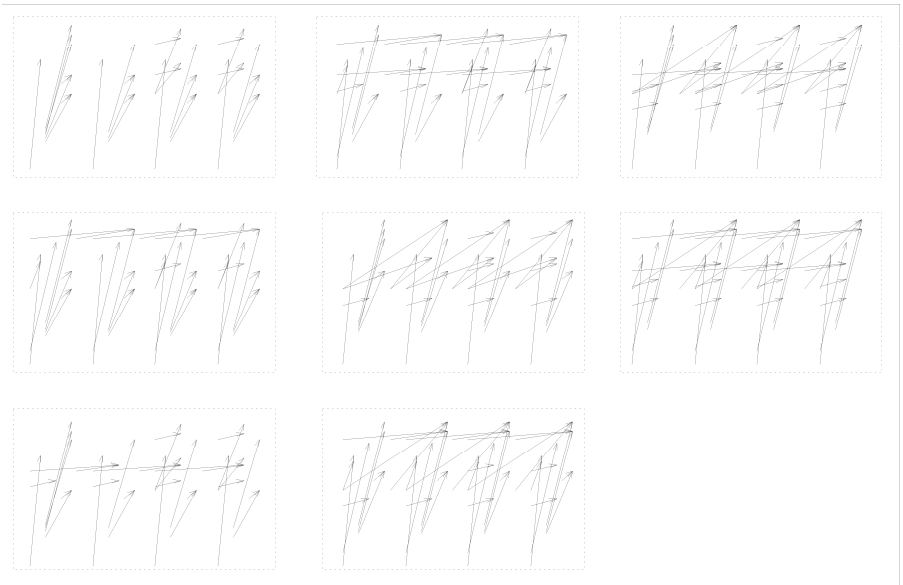
Although we cannot find a method of using these concepts in the inference, these results suggest the possibility of concept-generation using information other than rule number.



(a) Resulting Digraph (subgraph) with Rule



(b) Resulting Digraph (subgraph) with Data Attribute



(c) Resulting Digraph with Value

Fig. 18. Importance of color

4.5 Noise

Humans can compare the importance of individual data items, and learn new concepts even from noisy data by deleting the noise. All of the results shown above are simple noiseless examples, thus CLIP's ability to learning from noisy data remains in question. In theory, however, CLIP can learn new concepts from noisy data. The idea is illustrated in Fig. 19, where the circuit has the input going through a pass transistor (see hatched area of Fig. 19), and the

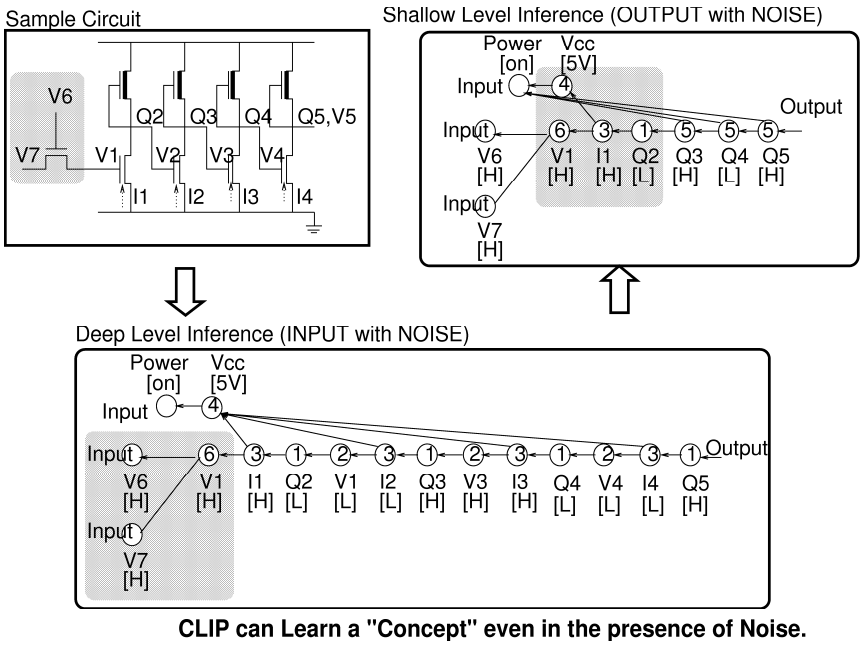


Fig. 19. Effect of noise

behavior of the pass transistor is simulated using rough knowledge about transistor behavior (Rule 6). The other inferences are performed using detailed knowledge (Rule 1, 2, 3 and 4), and the imbalance in the description levels makes a kind of noise in the inference process. [5]

Even in such a case, CLIP can learn concepts if the part of the inference affected by the noise is small enough. In other words, if the noise does not cause a large error in the graph size, CLIP can ignore the noise (*i.e.*, the hatched area of Fig. 19). If the noise affects the graph size, CLIP may learn somewhat different concepts (concepts affected by noise, or typical noise patterns themselves). However, if the environment is to be the same, the learned concepts will make the inference efficient.

Note that in the results shown in Figs. 15 and 16, some portion of the circuit behavior acts as noise in the learning process for *carry calculation*. For example, the outputs of the NOTs in Fig. 1 have no direct relation to the value of the carry. These results also indicate the learning potential of CLIP in noisy environments.

[5] Note that the value of V_1 in Fig. 3 is calculated from Q_1 using the detailed knowledge, *i.e.*, Rule 2, and Fig 3 does not involve this kind of noise. In this section, we examined the case where the occurrence of the inference that does not have a relationship with the target concepts makes the noise. Recent study [17] reveals the relationship between CLIP and the conventional inductive learning studies, which contributes to clarify the relation between CLIP and noise.

	COBWEB	CLIP
Input	• Set of object descriptions	• Set of **inference patterns** • Domain theory • Characteristics of inference engine
Output	• Useful sets of objects • Their conceptual descriptions	• Useful set of chunks **(composite objects)** from inference patterns • Conceptual descriptions to all chunks
Note	• Maximize category utility : to predict unknown properties	• Minimize graph size : to **improve efficiency**

Table 1
COBWEB and CLIP characteristics

5 Related Work

5.1 Inductive Learning

Most conventional inductive learning methods learn a new concept from positive/negative examples, and generate classification rules (*e.g.*, INDUCE, ID3 and Version Space [3]). They do not aim at to speed up abstract-level inference. CLIP assumes that *A concept is something which makes inference easy*, and tries to speed up the inference process by using the generated concepts. For example, CLIP generates new concepts such as NOT and NOR by analyzing the qualitative simulation traces of a digital circuit. CLIP can generate NOT and NOR without having any knowledge of logic.

Conceptual clustering [5] is an important area of machine learning in which several methods (*e.g.*, COBWEB) have been proposed to generate meaningful concepts from observations. CLIP is similar to COBWEB in that it finds useful concepts from observations by clustering similar patterns. Prior research in this area focused only on clustering similar patterns, and gave little attention to the inferences which use the learned concept. (See Table 1 for the difference.) The most important characteristic of CLIP is that it generates a set of rules to interpret the new concepts, and these interpretation rules enable the abstract-level inference using the generated concepts.

5.2 AM

AM [3] is an important concept-finding system. It uses a set of heuristics to find interesting concepts in mathematics, and can generate new theorems from the initial axioms. Although CLIP does not use an explicit heuristic, its minimization of the graph size has a heuristics aspect. CLIP has two important inputs, *i.e.*, the graph and the graph size definition. The former represents the characteristics of the environment, and the later, the characteristics of the inference system. CLIP tries to find concepts that make the inference efficient under the given environment.

We can see CLIP as a system which uses only one domain-independent heuristic: *A concept is something which makes inference easy.* The generality of this heuristic is one important characteristic of CLIP.

5.3 EBL

EBL (Explanation-Based Learning) [11] aims at improving problem-solving efficiency. Some research uses EBL to translate base-level descriptions of objects, such as *PART-OF(OBJ1,BOTTOM-1)*, *ISA(BOTTOM-1,BOTTOM)*, and *ISA(BOTTOM-1,FLAT)*, into abstract-level descriptions, such as *CUP(OBJ1)*.

Table 2 lists the features of EBL and CLIP for comparison. CLIP needs multiple examples, while conventional EBL requires only one example. Both need domain theory. CLIP also needs interpretation rules, *i.e.*, rules describing how to make an inference.

The main difference is that EBL needs a goal concept and operationality criteria to be explicitly provided as inputs. In CLIP, these are implicitly embedded in the algorithm and the multiple examples. CLIP can accommodate differences in the capability of each inference system with an appropriately defined graph size, which in turn affects the form of the extracted patterns.

EBL produces one macro-rule for a given goal concept, whereas CLIP produces a set of macros, which can be viewed as a set of concepts in some domains. CLIP also produces new interpretation rules that describe how to make an inference using the generated concepts.

CLIP shares an aspect of the EBL system. CLIP uses the graph size as a substitute for the Goal Concept and Operational Criteria to solve the utility problem [10]. Since most studies [8,6,10,13,14] related to the utility problem do not aim at generating new concepts, they lack a framework to perform inference by the generated concepts.

	EBL	CLIP
Input	• Goal concept • Training example • Domain theory • Operationality criteria	• **Multiple** training examples • Domain theory • **Characteristics of the inference engine**
Output	• One macro rule	• **Set of** macro rules **(operationality criterion** implicitly embedded) • **Interpretation rule :** (new domain theory for the abstract-level inference) • **Goal concept**

Table 2
EBL and CLIP characteristics

5.4 Genetic Algorithm

As mentioned in Section 2.2, the CLIP's pattern-finding algorithm can be seen as a kind of Genetic Algorithm [7]. One notable characteristic of the algorithm used in CLIP is the strong search ability of the pattern modification. Figure 20 shows how the graph size and the ratio of the views generated by pattern modification and by pattern combination changes with progressive iterations during the experiment shown in Fig. 16.

The Type I approximation is implemented as a kind of pattern modification by generating a temporary pattern which has a mark indicating the edge to be neglected. Neglecting some edge is equivalent to neglecting the connecting node color. Type II approximation is performed by neglecting some inputs when CLIP generates interpretation rules. When the approximation operation on a pattern results in contradiction, CLIP discards the view that contains the pattern. A solid circle on the top indicates such a discard. The solid curve indicates the average of the predicted graph sizes of all the candidate views. Note that the size is reduced to about one fifth of the original graph size.

In Fig. 20, the hatched area indicates the ratio of the views generated by pattern modification. As shown in the figure, CLIP's pattern modification with approximation included is the main source of the search. Improving the efficiency of the search is one important research theme of Genetic Algorithm. CLIP shares an aspect of the Genetic Algorithm. CLIP improves the search efficiency through the use of a special mutation operation, *i.e.*, pattern modification.

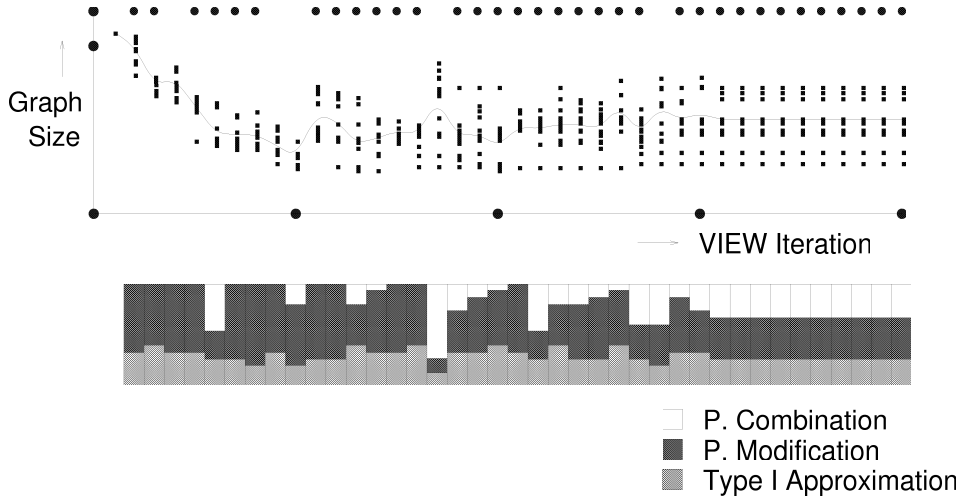


Fig. 20. Log of search (using approximation)

Note that the current CLIP implementation does not use a probabilistic framework. In this respect, CLIP is similar to the beam search programs, such as INDUCE[3] and HARPY[1].

5.5 Hierarchical Knowledge Representation

To understand a complex system, humans view a system at different levels of abstraction based on its functional structure. We have been developing a method to represent complex systems in a hierarchical manner [16,15].

The use of the entity-relationship model and the approximations is based on our previous research, and CLIP was designed to automate the process of hierarchically representing the functional descriptions.

Figure 21 shows our long-term research goal. We hope to integrate the hierarchical knowledge representation technique with the techniques for representing and acquiring the task knowledge.

6 Future Research Issues

This section summaries the remaining research issues.

Optimization of Multi-Level Representation: CLIP can generate a hierarchical concept structure such as that shown in Fig. 15 in a stepwise manner. However, CLIP doesn't have a mechanism that guarantees the optimality of the generated hierarchical structure. For example, CLIP may

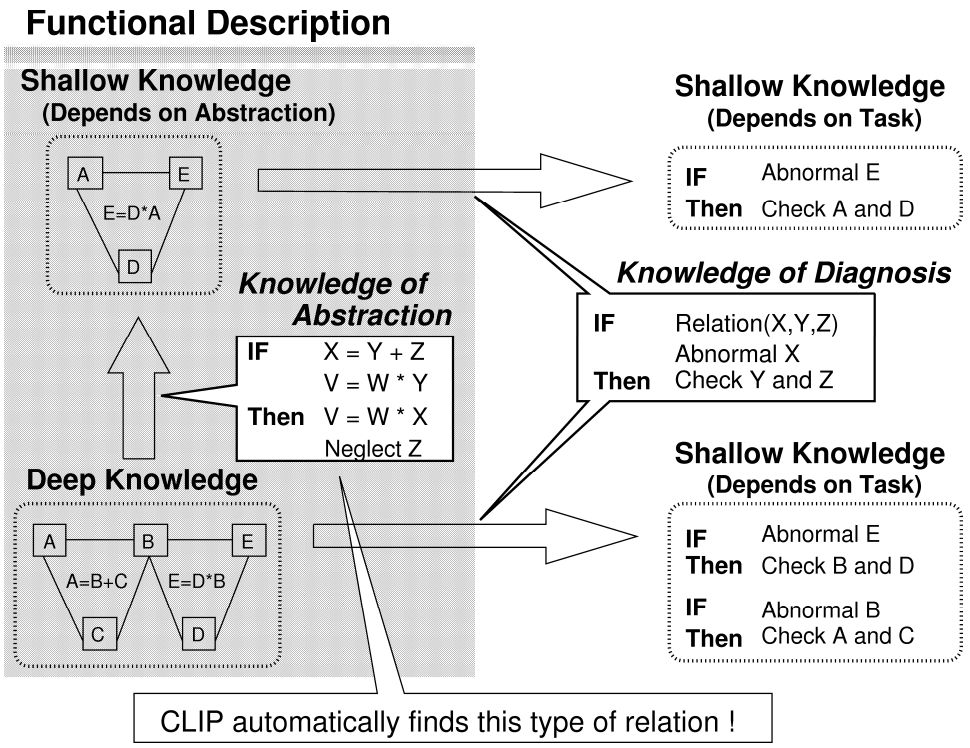


Fig. 21. Scope of hierarchical knowledge base

make two hierarchies when in fact three hierarchies would be optimal. Creating a mechanism to search for the optimum multi-level structure is one important future issue. Furthermore, as described in Section 5.5, CLIP generates a hierarchical functional description. How to integrate this description with the task knowledge and how the integration affects the hierarchy are also important issues.

Similarity to Human Concepts: The concepts generated in the experiments are all concepts known to be understood by humans. However, the reason why they correspond to human concepts is not clear. The graph size definition affects the generated concepts, and seems to be the key factor in this correspondence. Further analysis is necessary to confirm that the graph size definition is responsible for this correspondence.

Furthermore, the current implementation has some restriction on the form of the generated concepts. It can treat only a concept that has one output. Since the carry-chain circuit has two outputs, *i.e.*, the carry and the column value, the current program expresses this circuit as a combination of two independent concepts. The lack of multi-output concept treatment is another defect to be remedied.

Searching Ability: As described in Section 5.4, the current version of CLIP does not use a probabilistic framework, and this weakens the searching ability. For example, Fig. 22 shows how the rewritten graph size changes with successive iterations for the results shown in Fig. 11. At the leftmost white arrow in Fig. 22, CLIP first found a small pattern which resulted in a

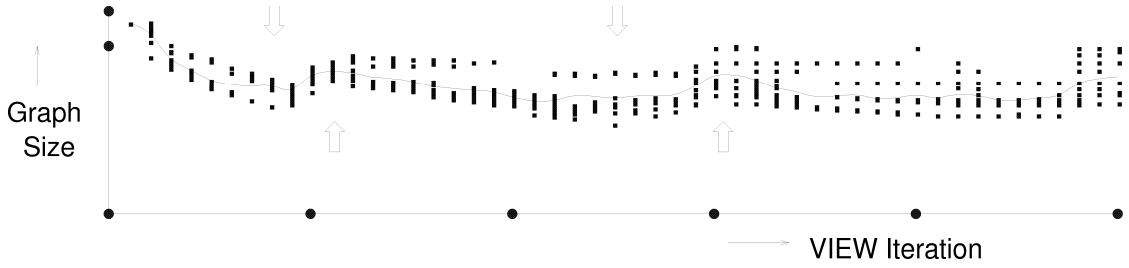


Fig. 22. Log of search (without approximation)

local minimum. Then the size increased until a larger pattern was found at the third arrow. The number of nodes in the pattern tends to increases with iteration, and CLIP tends to ignore other smaller patterns even if choosing one of these would result in a smaller graph. This is not a good aspect of CLIP's search ability, and needs improvement. [6]

7 Summary

CLIP is a system that extracts a set of concepts together with their interpretation rules from the inference traces produced during a problem-solving event. CLIP has now been fully implemented and tested. The results of our findings are summarized as follows:

- CLIP analyzes sample inference traces and extracts typical patterns which frequently appear in the inference. We assume that such patterns probably represent important concepts.
- CLIP generates a set of rules for interpreting new concepts, and these interpretation rules enable abstract-level inference.
- The environment and the inference system characteristics are the most important factors in constructing new concepts. Approximation is also important for creating more abstract-level concepts. The proposed method automatically considers these factors in generating new concepts.

In experiments, CLIP automatically generated multilevel representations of a carry-chain circuit from its given physical/single-level representations. The generated representations contain abstract-level concepts, including mathematical and logical concepts.

[6] Recent study [17] has enhanced the generality of both the representation capability of the digraph, and the algorithm for extracting typical patterns from the graph. In [17], we investigated how to apply CLIP to two seemingly different learning tasks: inductive learning of classification rules, and learning macro rules to speed up inference. This, in turn, has increased the importance of the theme of improving CLIP's search capability beyond the level assumed when we first wrote this paper.

Acknowledgement

The authors would like to thank Masaru Takeuchi for his helpful comments on the relation between CLIP and the Genetic Algorithm. They also extend their thanks to Chihiro Sugita for his help in implementing the CLIP program.

References

[1] A. Barr and E. A. Feigenbaum, editors. *The handbook of artificial intelligence*, chapter 5. William Kaufmann, Inc., 1981.

[2] P. P. Chen. The Entity-Relationship Model Towards a Unified View of Data. *ACM Transaction of Database Systems*, Vol. 1, No. 1, pp. 9–36, 1976.

[3] P. R. Cohen and E. A. Feigenbaum, editors. *The handbook of artificial intelligence*, chapter 12. William Kaufmann, Inc., 1982.

[4] J. de Kleer. A Qualitative Physics Based on Confluences. *Artificial Intelligence*, Vol. 24, pp. 7–83, 1984.

[5] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, Vol. 2, pp. 139–172, 1987.

[6] N. S. Flann. Applying Abstraction and Simplification to Learn in Intractable Domains. In *ML-90*, pp. 277–285, 1990.

[7] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, 1989.

[8] M. Lebowitz. Integrated Learning: Controlling Explanation. *Cognitive Science*, Vol. 10, pp. 219–240, 1986.

[9] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley Publishing Company, 1980.

[10] S. Minton. Quantitative Results Concerning the Utility of Explanation-Based Learning. In *AAAI-88*, pp. 564–569, 1988.

[11] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-Based Generalization : A Unifying View. *Machine Learning*, pp. 47–80, 1986.

[12] R. J. Mooney and S. W. Bennett. A Domain Independent Explanation-Based Generalizer. In *AAAI-86*, pp. 551–555, 1986.

[13] S. Yamada and S. Tsuji. Selective Learning of Macro-operators with Perfect Causality. In *IJCAI-89*, pp. 603–608, 1989.

[14] M. Yamamura and S. Kobayashi. An Augmentation of EBL on Plural Examples (in Japanese). *Journal of Japanese society for Artificial intelligence*, Vol. 4, No. 4,, 1989.

[15] K. Yoshida and H. Motoda. An Approach to Hierarchical Qualitative Reasoning (in Japanese). *Journal of Japanese society for Artificial intelligence*, Vol. 4, No. 4,, 1989.

[16] K. Yoshida and H. Motoda. Hierarchical Knowledge Representation based on Approximation. In *JKAW90*, pp. 345–360, 1990.

[17] K. Yoshida, H. Motoda, and N. Indurkhya. Graph-based Induction as a Unified Learning Framework. In *Applied Intelligence*, volume 4, pp. 297–328. Kluwer Academic Publishers, 1994.