ELSEVIER

Knowledge-Based Systems 9 (1996) 163–171

Knowledge-Based SYSTEMS

Efficient inference method for computing an optimal solution in predicate-logic hypothetical reasoning

Akiko Kondo[a], Mitsuru Ishizuka[b]

[a]*Multimedia Systems Laboratory, Fujitsu Laboratories Ltd, Nakahara-ku, Kawasaki 211, Japan*
[b]*Department of Information and Communication Engineering, Faculty of Engineering, University of Tokyo, Hongo, Bunkyo-ku, Tokyo 113, Japan*

Received 9 June 1994; revised 13 June 1995; accepted 26 June 1995

Abstract

Hypothetical reasoning, which is one type of abductive reasoning, is an important framework in the development of advanced knowledge-based systems. One problem with hypothetical reasoning is its slow inference speed, which is due to its nonmonotonic inference nature. A fast hypothetical reasoning system with predicate Horn clause expressions has been developed to overcome this problem. However, when the constraints for hypotheses are not strong, the number of hypotheses to be synthetized becomes too large to calculate. The paper presents an efficient hypothetical reasoning method combining best-first search, beam search and branch-and-bound search strategies for computing the optimal solution, which is the most desirable solution in many cases. The effectiveness of this method is shown experimentally using fault-diagnosis problems in logic circuits.

Keywords: Hypothetical reasoning; Predicate logic; Optimal solution search

1. Introduction

Knowledge is often incomplete; that is, it often involves exception or contradiction. The handling of incomplete knowledge in knowledge bases is an important function in expanding the capability of current knowledge bases. Hypothetical reasoning handles such incomplete knowledge as hypothesis [1,2]. It is one type of abductive reasoning, and it can be directly applied to model-based diagnosis problems [1,3], design problems [4] etc. Thus hypothetical reasoning is an important framework in the development of advanced knowledge-based systems from both the theoretical and practical viewpoints.

One crucial problem with hypothetical reasoning is, however, its slow speed, which is due to its nonmonotonic reasoning nature. We have developed fast hypothetical reasoning systems to overcome this problem [5,6]. KICK-HOPE (Knowledge-base handling InComplete Knowledge – by HOlding Parallel solutions on Environment lattice) [6] is one of those systems. Its reasoning mechanism is related to the SLD-AL [7] or QSQR [8] methods developed as efficient deductive database technology, and it works for predicate Horn-clause knowledge.

KICK-HOPE shows high efficiency for many problems. However, when the constraint for hypotheses is not strong, the number of hypotheses to be synthetized becomes too large to calculate and the inference speed goes down. This problem is inevitable because KICK-HOPE calculates all the solutions as in database applications. In many practical cases, we need one or a few good solutions quickly rather than all the solutions. Therefore, we present here a fast hypothetical reasoning method based on KICK-HOPE that obtains an optimal solution by combining best-first search, beam search and branch-and-bound search strategies. The effectiveness of this method is shown experimentally using fault-diagnosis problems in logic circuits.

2. Logic-based hypothetical reasoning system

Our hypothetical reasoning is based on a logic framework used in Theorist [1,2], in which knowledge is classified into fact (background knowledge in our system) or hypothesis. Fact, or background knowledge, is knowledge which is always correct and has no possibility of contradiction. On the other hand, hypothesis is defeasible knowledge having the possibility of

SSDI 0950-7051(95)01015-7

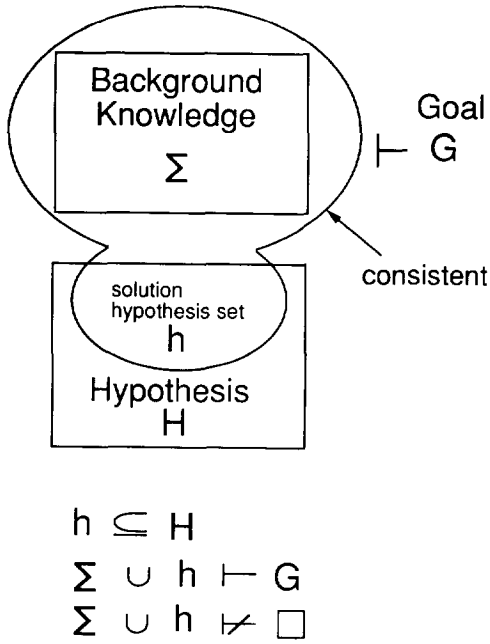

Fig. 1. Basic structure of logic-based hypothetical reasoning system.

contradiction by other knowledge. The basic behaviour of this hypothetical reasoning can be described as follows: if a given goal cannot be proved with only background knowledge, the reasoning system adopts consistent hypotheses necessary to prove the goal.

The aim of this hypothetical reasoning is to obtain these consistent hypotheses, which we call 'solution hypotheses'. In general, a deductive-inference mechanism can be used for this hypothesis generation. Hypothetical reasoning can be called abductive reasoning especially when hypotheses are abduced from a knowledge base. The hypothetical reasoning can be applied to many practical problems. For example, its solution hypothesis becomes the representation of faults in a diagnosis problem [3], the combinations of possible design components in a design problem [4] etc.

Fig. 1 shows the basic structure of our hypothetical reasoning. The knowledge base consists of two parts. One is a set of background knowledge Σ (which is always true in any environment) and the other is a set of hypotheses H (which are not always true in an environment and sometimes contradict other knowledge). Let G be a given goal, and h be a subset of H. Then the basic function of this system is to find a solution hypothesis h satisfying the following three logical formulae:

$h \subseteq H$

(h is a subset of H),

$\sum \cup h \vdash G$

(G is proved from Σ and h),

$\sum \cup h \nvdash \square$

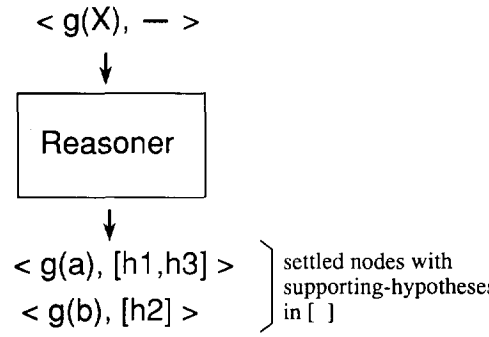

Fig. 2. Function of reasoner in KICK-HOPE.

(Σ and h are consistent). There exist plural solution hypotheses hs in general that satisfy these conditions. 'All-solution search' means the obtaining of all the feasible solution hypotheses. 'Optimal search' means the obtaining of at least one solution hypothesis such that its evaluation value is minimal (or maximal).

3. Kick-Hope: a fast hypothetical reasoning system for predicate-logic knowledge bases

One crucial problem with hypothetical reasoning is its slow inference speed, which is due to its nonmonotonic nature. One way to overcome this problem is to combine the advantages of the forward and backward reasoning styles. Forward reasoning enables us to avoid the recalculation of the same inference tree, and backward reasoning allows a restricted search only for a goal-related inference tree. KICK-HOPE [5] has basically incorporated these advantages. We first describe KICK-HOPE's mechanism briefly, since the inference mechanism of this paper is based on it. We use Prolog notation, i.e. a variable starts with a capital letter, such as X, and a constant starts with a lower-case letter, such as a.

Fig. 2 illustrates the reasoner's function for KICK-HOPE. We obtain two kinds of information for a node through the reasoner, i.e. restricted values for variables in the node and necessary hypotheses to prove the node. We call an output node of the reasoner a 'settled-node', and hypotheses in the settled node 'supporting hypotheses'.

Fig. 3 shows an example of the reasoner's operation in KICK-HOPE. When a node $g(X)$ is put into the reasoner, its child nodes $g1(X)$ and $g2(X)$ are generated at first. Then they are put into other reasoners sequentially from the left node in order. Since $g1(X)$ and $g2(X)$ have an and relation, the restriction between their variables should be considered. After the generation of a settled node for $g1(X)$, the variable X is restricted to a. As a result, $g2(a)$, instead of $g2(X)$, is put into the reasoner. This avoids the generation of unnecessary settled nodes ($\langle g2(b), [h4] \rangle$ in this example). After all

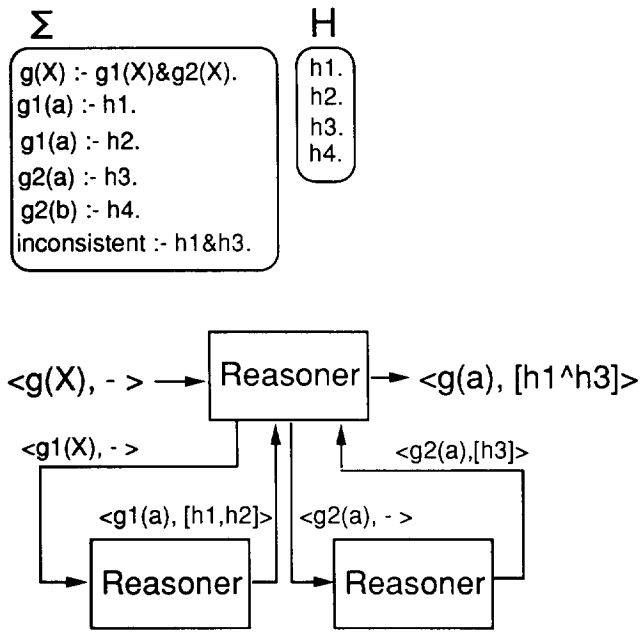

Fig. 3. Example of knowledge base and reasoner's operation in KICK-HOPE.

the child nodes are settled, their supporting hypotheses are synthetized to generate $g(X)$'s settled-node $\langle g(a), [h1 \wedge h3]\rangle$. Here, h1 $\wedge$ h3 implies a synthetized-hypothesis, whose synthesis is carried out by the bit wise or operation of the bit vectors of $h1$ and $h3$. This hypothesis-synthesis operation is based on ATMS [9], and it allows efficient calculation of the minimal supporting hypothesis by the use of a hypothesis lattice structure.

In this method, backward reasoning is adopted for generating necessary child nodes, and forward reasoning is adopted for the hypothesis synthesis on the hypothesis lattice. With these mechanisms, KICK-HOPE has a high efficiency in comparison with simple reasoning based on a simple inference mechanism embedded in Prolog.

However, when the constraints for hypotheses are not strong, the number of supporting hypotheses becomes large. Since the cost of the hypothesis synthesis which corresponds to join operations in relational databases is very expensive, the reasoning speed slows down accordingly. This problem is mainly due to KICK-HOPE's mechanism of calculating all the solutions.

Thus, we have developed a fast hypothetical reasoning system KICK-HOPE II for obtaining optimal solutions instead of all the solutions. The efficiency of this system is achieved primarily by reducing the number of hypothesis-synthesis operations. We describe its reasoning mechanisms in the next section.

4. Optimal search for hypothetical reasoning

In this section we describe optimal search strategies compatible with KICK-HOPE. KICK-HOPE obtains all the supporting hypotheses for a settled node as shown in Fig. 2. However, not all the supporting hypotheses for every node are necessarily required to obtain the optimal solution for a given goal. In optimal search, only an optimal supporting hypothesis has to be generated at first for each node. In subsequent reasoning, when an optimal supporting hypothesis is judged to be unsuitable, a second optimal supporting hypothesis has to be generated. This case occurs when

- this optimal supporting hypothesis contradicts the supporting hypothesis of the other node, or
- it is possible for another supporting hypothesis (not the optimal one) to compose an optimal solution for the goal.

Considering these situations, we can reduce the number of hypothesis-synthesis operations to obtain an optimal solution effectively.

For the evaluation of optimality, we assign a numerical weight to each element hypothesis. An optimal solution is a supporting hypothesis for which the weight sum of the element hypotheses is minimal. We describe below optimal search strategies applied for hypothesis synthesis that utilize the weight of the hypothesis. If, in this paper, we consider the weight to be the cost of the hypothesis, this type of reasoning can be called cost-based hypothetical reasoning, as seen, for example, in [10].

4.1. Weight of hypothesis and lower bound of unsynthetized hypothesis

We define here the weight of an element hypothesis as an integer larger than 0 and the weight of a hypothesis as the weight sum of its element hypotheses. An optimal solution is a supporting hypothesis for which the weight is minimal for a given goal.

Thus a light-weighted hypothesis has priority over a heavy one. In a case in which the weight of an element hypothesis is not clearly defined, its weight is considered to be 1 (the default value). If the weight of every element hypothesis is 1, a hypothesis with a small number of element hypotheses has priority, because the weight of a hypothesis coincides with the number of its element hypotheses.

Next we consider the lower bounds (LBs) of unsynthetized hypotheses (hypotheses which are not yet synthetized). We effectively use this value to decide the order of merge operations (for hypothesis synthesis). The LB of an unsynthetized hypothesis has a value such that the weight of the hypothesis always becomes more than or equal to this value after synthesis.

We define the LB of an unsynthetized hypothesis as follows:

- The LB of $A\&B$ is the maximum of A's LB or B's LB, and
- The LB of (A or B) is the minimum of A's LB or B's LB.

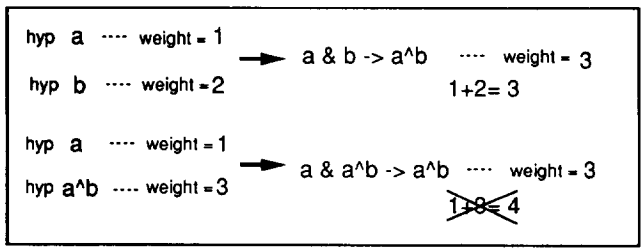

Fig. 4. Examples of calculation of weight of synthetized hypothesis.

Fig. 4 shows the reason why the LB of $A\&B$ is not simply the sum of A's LB and B's LB. When the weight of hypothesis a is 1 and that of hypothesis b is 2, the weight of hypothesis $a \wedge b$ becomes 3 ($1 + 2 = 3$). However, the weight of hypothesis $a\&(a \wedge b)$ becomes 3 (not $1 + 3 = 4$), because $a\&(a \wedge b)$ becomes $a \wedge b$ after synthesis. Thus, in the case in which one hypothesis is included in the other one, or both hypotheses have common elements, the weight of the synthetized hypothesis does not simply become the sum of each weight. As the synthetized-hypothesis weight becomes minimal when one hypothesis is included in the other one, the maximum LB of the component hypotheses becomes the LB of the unsynthetized hypothesis.

On the other hand, it is natural to define the LB of (A or B) as the minimum of A's LB and B's LB.

4.2. Optimal search strategies for hypothesis synthesis

In this section, we describe optimal search strategies for hypothesis synthesis using the weights and LBs of unsynthetized hypotheses.

We utilize the following three optimal search strategies for efficient hypothesis synthesis:

- best-first search,
- beam search,
- branch-and-bound search.

We describe these mechanisms briefly, and a way in which they can be applied to our hypothetical reasoning with reference to the example of Fig. 5.

4.2.1. Best-first search

This is a strategy [11] of searching from the best-valued node in order. We utilize this strategy to decide the order of synthetizing hypotheses. In many cases, the evaluation for deciding the order has been made on the basis of heuristic knowledge. However, since we can easily compute the LBs of subgoals in logic-based hypothetical reasoning as described in the previous section, we use these LBs for the evaluation without any other heuristic knowledge. Since the LB changes with the process of hypothesis synthesis, the updated LB must be used. In the example of Fig. 5, as the LB of $sg1$ is 1 and that of $sg2$ is 2, the supporting hypotheses for $sg1$ are synthetized first. If the LB of $sg1$ exceeds that of $sg2$ in the process of hypothesis synthesis in $sg1$, it is interrupted, and hypothesis synthesis for $sg2$ starts.

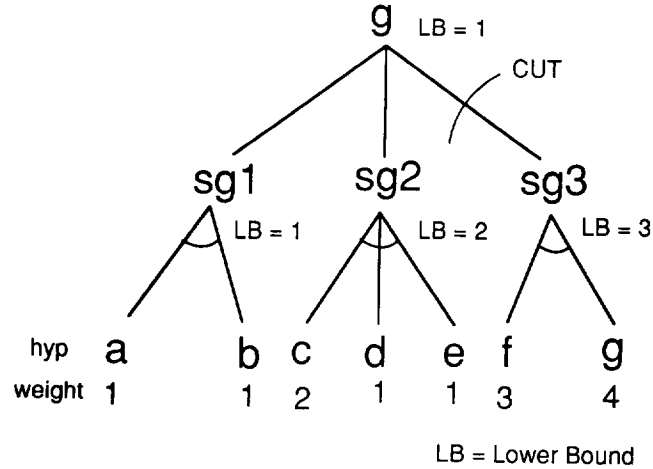

Fig. 5. Example of lower bound for nodes in inference tree.

Fortunately, it is guaranteed that the optimal solution can be obtained when the LB is used for the evaluation in the best-first search strategy. Accordingly, solutions obtained in our system are always optimal ones. The best-first search strategy generally has the drawback of requiring a huge memory area. We can, however, reduce this problem by using the following beam-search strategy.

4.2.2. Beam search

This is strategy [11] of restricting the hypotheses under consideration to only promising ones in the middle of the search. There are two kinds of strategies for the selection of hypotheses. One is to select a certain number of good-evaluated hypotheses (this number is called the beam width) and remove the others. The other strategy is to remove the hypotheses whose weights have a larger value than a certain threshold, and select the remaining ones.

In hypothetical reasoning, this beam-search strategy does not guarantee the optimality of the solutions. This is because the remaining hypotheses in the intermediate nodes possibly become inconsistent in the subsequent reasoning. In particular, when the beam width is used to select the hypotheses, the necessary hypotheses are often removed in the intermediate nodes. On the other hand, selecting hypotheses by a threshold is useful in the case in which we do not need hypotheses that are weighted too heavily. When a value is set up as a threshold weight, it behaves as a global constraint in the reasoning. With this strategy, if solutions are obtained, they are always optimal ones. If no solution is obtained, the threshold weight should be increased because it is too small to obtain solutions.

Thus we adopt the beam-search strategy with a threshold in our hypothetical reasoning. In Fig. 5, when the threshold weight is set to 3, subgoal $sg3$ is removed because the LB of $sg3$ becomes 4.

This beam-search strategy should be used secondarily in order to compensate for the problem of the best-first search strategy, i.e. the consumption of a huge memory area.

4.2.3. Branch-and-bound search

This strategy [12,13] generates one solution at first and memorizes its weight as a temporary weight. Then it prunes the branches whose weights exceed this value, and the temporary weight is updated when a better-valued (lighter-weighted) solution is obtained.

In the example of Fig. 5, as the supporting hypothesis of $sg1$ is $a \wedge b$ and its weight is 2, the temporary weight is set to 2. In the process of hypothesis synthesis in $sg2$, the LB of $sg2$ exceeds 2. That is, when hypothesis c and hypothesis d are combined, the weight becomes 3. Since it becomes certain that the weight of $sg2$ is larger than 3, the hypothesis synthesis in $sg2$ becomes unnecessary.

We utilize this strategy for unsettled or relation nodes. When optimal supporting hypotheses for settled nodes are obtained, their smallest weight is propagated to other or-relation nodes as a temporary weight. Thanks to this procedure, it becomes unnecessary to obtain the optimal supporting hypothesis for every node. In other words, we do not need to obtain the optimal supporting hypothesis for a node for which the LB exceeds the temporary weight.

In general, this branch-and-bound search strategy can be adopted only for the final goal in hypothetical reasoning, because, even if an optimal supporting hypothesis is obtained in an intermediate node, it does not always become a component of the optimal solution for a given goal. Also, the optimal supporting hypothesis in the intermediate node may contradict other hypotheses to be combined in the subsequent reasoning. Thus we also keep nonoptimal hypotheses (we call them next candidate hypotheses) with optimal ones in the intermediate node, and use them when needed in the subsequent reasoning. The next-candidate hypotheses consist of unsynthetized hypotheses and synthetized hypotheses which are not optimal at that time.

We have incorporated these strategies into the hypothesis synthesis in KICK-HOPE, and have developed KICK-HOPE II, a fast predicate-logic hypothetical reasoning system for computing an optimal solution. The data structure and the reasoning algorithm of KICK-HOPE II are described in some detail in the next section for those who need to know them precisely, for example to implement a similar mechanism.

5. Kick-Hope II: a fast predicate-logic hypothetical reasoning system for computing an optimal solution

5.1. Data structure

In general, rule-type hypotheses are allowed. However,

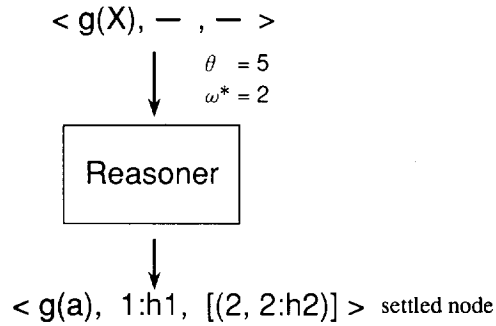

Fig. 6. Function of reasoner in KICK-HOPE II.

using the same transformation as that used in KICK-HOPE [6] etc. [5], we can transform them into unit-clause (fact-type) hypotheses and rule-type background knowledge by preprocessing. As in KICK-HOPE, we deal with the case in which hypotheses are defined as grounded unit clauses not including variables. In this case, the number of possible hypotheses becomes finite, and they can be expressed as bit vectors, allowing efficient reasoning operations, particularly for hypothesis synthesis and consistency checking as in ATMS [9]. A weight value (which is an integer larger than 0) is set for each hypothesis.

The data structure of nodes in KICK-HOPE II is of the form of

⟨Node-name, Optimal-supporting-hypothesis, Next-candidate-hypothesis⟩

Here, Optimal-supporting-hypothesis is denoted as *Weight: Hypotheses*, and Next-candidate-hypothesis is expressed as a list, the elements of which are (*LB, Non-optimal-supporting hypothesis*), and are sorted by their LBs. At the initial stage, Node-name may have variables, and Optimal-supporting-hypothesis and Next-candidate hypothesis are undecided. When the nodes are put into the reasoner of KICK-HOPE II with a temporary weight ω^* and a threshold weight θ, they are transformed into settled nodes (see the example of Fig. 6).

The settled nodes are classified into the following four categories (each Node-name is instantiated):

- *True node (always true) (no supporting hypothesis is required):*

  ⟨Node-name, true, –⟩

- *False node (always false):*

  ⟨Node-name, false, –⟩

- *Node whose optimal supporting hypothesis is determined:*

  ⟨Node-name, Optimal-supporting-hypothesis, Next-candidate-hypothesis⟩

- *Node where there exists no lighter-weighted supporting hypothesis than a temporary weight (in this case an*

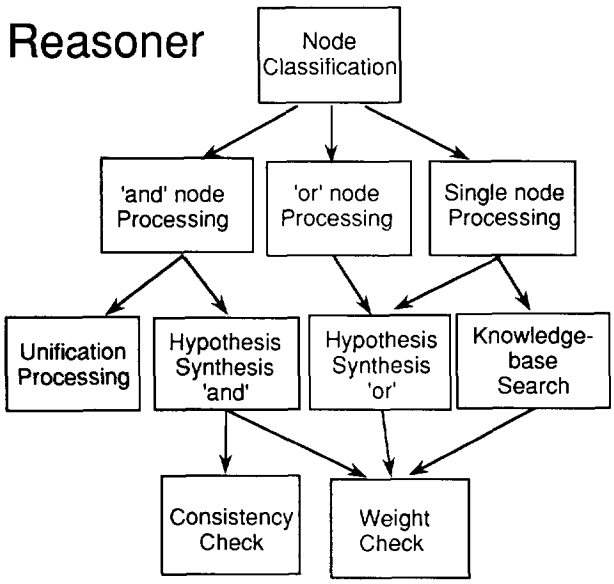

Fig. 7. Components of reasoner in KICK-HOPE II.

optimal supporting hypothesis does not need to be obtained):

⟨Node-name, 'non', Next-candidate-hypothesis⟩

The fourth type of node appears when the LB of the node exceeds the temporary weight ω^* in the process of generating its optimal supporting hypothesis. In any case, its next-candidate hypothesis must be kept because the next-candidate hypothesis is occasionally used in the reasoning for upper nodes. (The supporting hypotheses of the upper (or parent) nodes are synthetized from the supporting hypotheses of their lower (or child) nodes.)

5.2. Reasoning algorithm

Fig. 7 shows the reasoner operation of KICK-HOPE II. At first a node is judged to be one of the following: a 'single node' (which consists of one literal), an 'and node' (which is a complex node having an and relationship), or an 'or node' (which is a complex node having an or relationship). If the node is a single node, 'single-node processing' is executed. If the node is an and node, 'and-node processing' is executed. Otherwise, 'or-node processing' is executed[1].

We give algorithms for these processings with the temporary weight ω^* and the threshold weight θ for a node.

Algorithm for single node N:

(1) Generate settled nodes and update ω^* in accordance with the following four cases:

(1.1) *Case in which N is unified with fact-type (unit-clause) background knowledge:* Generate a true node $\langle N, \text{true}, - \rangle$, and set ω^* to 0.

(1.2) *Case in which N is unified with the hypothesis:* Let H and ω indicate the unified hypothesis and the weight of H, respectively. Then,

- *beam search:* if $\omega > \theta$, generate nothing
- *branch-and-bound search:* if $\omega > \omega^*$, generate $\langle N, \text{non}, [(\omega, \omega : H)] \rangle$.

Otherwise, generate $\langle N, \omega : H, [\,] \rangle$, and set ω^* to ω.

(1.3) *Case in which N is unified with the head of rule-type background knowledge:* Let B indicate the body of unified rule-type background knowledge. Then, put the node $\langle B, -, - \rangle$ into the reasoner with ω^* and θ to generate settled nodes. Remove the false nodes among these settled nodes. Rewrite the first terms (Node-name) of all the remaining settled nodes as N, whose variables are substituted according to the unified information.

(1.4) *Case in which N cannot be unified with any knowledge:* Generate a false node $\langle N, \text{false}, - \rangle$, and terminate.

(2) Merge the settled nodes which have the same unified constants in N into one node by synthetizing their supporting hypotheses through a 'hypothesis-synthesis or' operation, to be described in the next section. If no settled node is obtained, generate a false node $\langle N, \text{false}, - \rangle$.

Algorithm for and node N1 & N2:

(1) Generate settled nodes $SN1$ for $N1$ through the reasoner with ω^* and θ.

(2) In the case in which there is no element in $SN1$, or $SN1$ is a false node, terminate. In the case in which no settled node for $N1\&N2$ is obtained, generate a false node $\langle N1\&N2 \text{ false}, - \rangle$, and terminate.

(3) Take one node $FN1$ from $SN1$ and denote the remaining nodes as $SN1'$. Also, let $N2'$ be an $N2$ whose variables are substituted according to the unified information of $FN1$.

(4) Generate settled nodes $SN2$ for $N2'$ through the reasoner with ω^* and θ.

(5) In the case in which there is no element in $SN2$, or $SN2$ is a false node, rewrite $SN1'$ to $SN1$ and go to step 2.

[1] A complex node (and node/or node) having more than two predicates is processed from the left operator. For example, *A&B&C* is processed as an and node for *A* for *B&C*.

(6) Take one node $FN2$ from $SN2$, and denote the remaining nodes as $SN2'$.

(7) Synthetize the supporting hypotheses in $FN1$ and $FN2$ through a 'hypothesis-synthesis-and' operation, to be described in the next section. In the case in which synthetized hypotheses are obtained, generate a settled node whose Node-name is $FN1\&FN2$, and update ω^*. Rewrite $SN2'$ as $SN2$ and go to step 5.

Algorithm for or node N1 or N2:

(1) Generate settled nodes $SN1$ for $N1$ through the reasoner with ω^* and θ, and update ω^*. Remove false nodes among these settled nodes.

(2) Generate settled nodes $SN2$ for $N2$ through the reasoner with ω^* and θ, and update ω^*. Remove false nodes among these settled nodes.

(3) Merge the nodes in whose arguments the predicates and constants are exactly the same onto one node by synthetizing their supporting hypotheses through a hypothesis-synthesis or operation, to be described in the next section. If no settled node is obtained, generate a false node $\langle N1$ or $N2$, false, $-\rangle$.

5.3. Hypothesis synthesis

A and B in the algorithms of this section are 'true' or unsynthetized hypotheses. The symbol $^{++}$ in the algorithms indicates the operation of obtaining the optimal hypothesis from an unsynthetized hypothesis. This can be calculated by combining 'hypothesis-synthesis and' and 'hypothesis-synthesis or' operations, to be described below.

Algorithm for hypothesis synthesis A&B:

(1) If both A and B are true, generate 'true'.
(2) If A is true, generate an optimal supporting hypothesis and a next-candidate hypothesis from B^{++}.
(3) If B is true, generate an optimal supporting hypothesis and a next-candidate hypothesis from A^{++}.
(4) Synthetize the elements in A and B to generate an optimal supporting hypothesis and a next-candidate hypothesis as described below.
  (4.1) Generate an optimal supporting hypothesis A_1 and a next-candidate hypothesis A_OTH from A^{++}. Also, generate an optimal supporting hypothesis B_1 and a next-candidate hypothesis B_OTH from B^{++}. Regarding $A\&B$ as the aggregation of $A_1\&B_1$, $A_1\&B\_OTH$, $A\_OTH\&B_1$ and $A\_OTH\&B\_OTH$, sort them according to their LBs for best-first search.
  (4.2) Synthetize hypotheses in order. If a synthetized hypothesis is inconsistent, or its weight exceeds θ, remove this hypothesis for beam search.
  (4.3) Terminate the hypothesis synthesis when the LB of every unsynthetized hypothesis exceeds ω^*. In the case in which a hypothesis with a minimal weight of less than ω^* is obtained, set this as an optimal supporting hypothesis. Otherwise, set non instead of optimal supporting hypothesis. Sort the unsynthetized hypotheses and nonoptimal synthetized hypotheses according to their LBs, and set them as a next-candidate-hypothesis for branch-and-bound search.
  (4.4) In the case in which no synthetized hypothesis is obtained after synthetizing every element, generate nothing (this means the failure of the hypothesis synthesis.).

Algorithm for hypothesis synthesis A or B:

(1) If either A or B is true, generate 'true'.

(2) Synthetize the elements in A and B^{++} in order, while sorting them every time their LBs are updated for best-first search.

6. Experimental evaluation of inference speed

We evaluate the inference speed of KICK-HOPE II using the example of fault-diagnosis problems for logic circuits. We define the state of each gate ('normal', 'stuck-on' or 'stuck-off') as hypotheses. In this example, the number of hypotheses is in proportion to the number of gates. Therefore, in large-scale circuits, determining fault gates from observations becomes very time-consuming, as the number of possible combinations of hypotheses (the states of gates) increases exponentially. KICK-HOPE II effectively solves this problem by setting up weights for hypotheses. That is, by setting the weight to

- 1 for the case in which a gate is 'normal',
- N for the case in which a gate is abnormal ('stuck-on' or 'stuck-off') (where N is an adequately large integer),

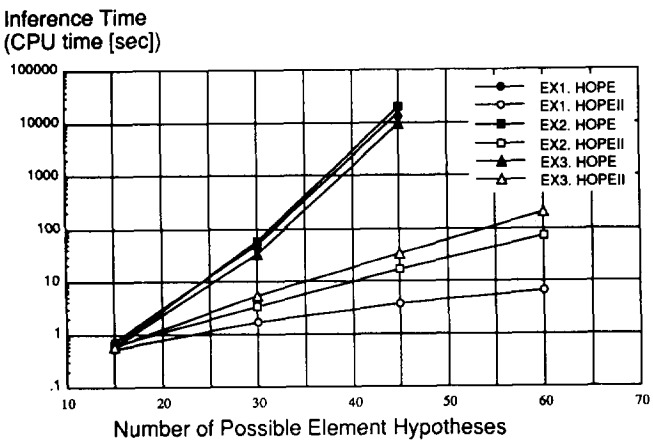

Fig. 8. Experimental results for inference times for fault diagnosis problems in adder circuits.

Table 1
Number of generated solutions in fault diagnosis problems in adder circuits

Number of hypotheses		Number of generated solutions			
		15	30	45	60
Example Ex 1	HOPE	19	453	11351	–
	HOPE II	1	1	1	1
Example Ex 2	HOPE	14	498	13006	–
	HOPE II	2	1	1	1
Example Ex 3	HOPE	14	356	13028	–
	HOPE II	1	1	1	1

the problem can be solved before calculating all the combinations, since the 'normal' state hypothesis has priority during the inference. Practically, calculating all the combinations is not smart, because the number of fault gates is one or a very few in many cases. As N is larger, the normal state hypothesis has higher priority. In this example, we set N to be around the number of gates.

Furthermore, KICK-HOPE II utilizes a threshold weight to limit the solution hypothesis weight. In this example, we set it so that the number of fault gates becomes less than four.

Fig. 8 shows the inference times, and Table 1 gives the number of solutions obtained by KICK-HOPE II and KICK-HOPE for this example. EX1, EX2 and EX3 in Table 1 and Fig. 8 are problems with different types of faults. It is shown that the inference speed of KICK-HOPE II is improved to a great extent because of a small number of generated solutions.

The number of fault gates in the optimal solution is zero in EX1, one in EX2, and two in EX3. Fig. 8 and Table 1 show that the inference time becomes longer as the number of fault gates increases. However, the reasoning is expected to compute the solution in a short time in most cases, because it is very rare for many gates to behave abnormally at the same time.

In the case in which the optimal solution-hypothesis output does not include correct fault gates (for example, in the case in which the optimal solution indicates one fault, in spite of their being two fault gates in the circuit), KICK-HOPE II can easily compute the second optimal solution because the next-candidate hypothesis is always kept at every node in the system.

7. Conclusions

We have described a fast hypothetical reasoning system called KICK-HOPE II that computes an optimal solution from a predicate-logic knowledge base and a given goal. KICK-HOPE II has overcome KICK-HOPE's problem, i.e., the increase in the number of hypothesis-synthesis operations, by incorporating best-first search, beam search and branch-and-bound search strategies to obtain an optimal solution.

However, in the case in which the constraints for the hypotheses are strong enough, the efficiency of KICK-HOPE is higher than that of KICK-HOPE II. This is because the number of solutions does not become so large in both systems, whereas KICK-HOPE II executes more complicated operations for weights than does KICK-HOPE. Thus, it can be said that KICK-HOPE II is useful for the case in which an optimal solution is required, or the number of possible solutions is supposed to become large because of weak hypothesis constraints.

The inference time of KICK-HOPE II is influenced by the weights of the element hypotheses. In the case in which differences between the weights are large, high efficiency can be expected in general. On the other hand, in the case in which differences between the weights are small, high efficiency cannot be expected, because of the decrease in the priority-search effect. However, even in the latter case, the reasoning can be accomplished before calculating all the combinations, because the number of hypotheses in an intermediate node may vary.

The AAA/H algorithm by Ng and Mooney [14] has an objective similar to that of our system. That algorithm uses a method of setting a beam width for the beam-search strategy. The strategy of using a beam width, however, does not guarantee the optimality of the solutions. Furthermore, it is risky to adopt beam search as a main strategy in hypothetical reasoning, because the risk is high that all the synthetized hypotheses in an intermediate node will eventually become inconsistent in the subsequent reasoning.

Since the computational complexity of nonmonotonic reasoning, including hypothetical reasoning, has been proved to be NP-complete or NP-hard [15,16], KICK-HOPE II cannot exceed the limit of exponential-order inference times with respect to the problem size as shown in Fig. 8 and Table 1. To overcome this problem, approximate methods for computing a near-optimal

solution [17], reasoning with a learning or analogy mechanism [18,19], and knowledge-base compilation [20] are promising approaches.

Acknowledgements

This work was supported by the Japanese Ministry of Education Grant-in-Aids 04452190 (B) and 04229105 (special area on knowledge science).

References

[1] D. Poole, R. Aleliunas and R. Goebel, Theorist: a logical reasoning system for defaults and diagnosis, in N.J. Cercone and G. McCalla (eds.), The Knowledge Frontier: Essays in Knowledge Representation: Springer-Verlag, USA, 1987, pp. 331–352.

[2] D. Poole, A logical framework for default reasoning. Artif. Intell., 36 (1988) 27–47.

[3] M. Ishizuka and T. Matsuda, Knowledge acquisition mechanisms for a logical knowledge base including hypothesis, Knowledge-Based Systems, 3 (1990) 77–86.

[4] T. Makino and M. Ishizuka, A hypothetical reasoning system with constraint handling mechanism and its application to circuit-block synthesis, in Proc. PRICA '90, Nagoya, Japan, 1990, pp. 122–127.

[5] F. Ito and M. Ishizuka, Fast hypothetical reasoning system using inference-path network, J. JSAI, 6 (1991) 501–509 (also Proc. TAI '91, San Jose, CA, USA, 1991, pp. 352–359).

[6] A. Kondo, T. Makino and M. Ishizuka, An efficient inference for hypothetical reasoning system for predicate-logic knowledge-base, in Proc. TAI '91, San Jose, CA, USA, 1991, pp. 360–367 (also Efficient hypothetical reasoning system for predicate-logic knowledge base, Knowledge-Based Systems, 6 (1993) 87–94).

[7] L. Vieille, A database-complete proof procedure based on SLD-resolution, in Proc. 4th ICLP, 1987, pp. 74–103.

[8] L. Vieille, From QSQ towards QoSaQ: global optimization of recursive queries, in Proc. Expert Database Systems, 1988, pp. 421–435.

[9] J. de Kleer, An assumption-based TMS, Artif. Intell., 28 (1986) 127–162.

[10] E. Charniak and S. Shimony, Probabilistic semantics for cost based abduction, in Proc. AAAI '90, 1990.

[11] A. Bar and E.A. Feigenbaum (eds.), The Handbook of Artificial Intelligence, Vol. 1, William Kaufmann, USA, 1981.

[12] E.L. Lawer and D.E. Wood, Branch-and-bound methods: a survey, Operations Research, 14 (1966) 699–719.

[13] T. Ibaraki, Enumerative approaches to combinatorial optimization, Annals of Operations Research, 10 and 11 (1987).

[14] H.T. Ng and R.J. Mooney, An efficient first-order Horn-clause abduction system based on the ATMS, in Proc. AAAI-91, 1991, pp. 494–499.

[15] H.A. Kautz and B. Selman, Hard problems for simple default logics, in Proc. KR '89, Toronto, Canada, 1989, pp. 189–197.

[16] J. Stillman, It's not my default: the complexity of membership problems in restricted propositional default logics, in Proc. AAAI-90, 1990, pp. 571–578.

[17] M. Ishizuka and T. Okamoto, A polynomial-time hypothetical reasoning employing an approximate solution method of 0–1 integer programming for computing near-optimal solution, in Proc. Canadian Conf. AI, Banff. Canada, 1994, pp. 179–186.

[18] T. Makino and M. Ishizuka, Speedup of hypothetical reasoning by experience-based learning mechanism, Knowledge-Based Systems, 7 (1994) 189–198.

[19] M. Ishizuka and A. Abe, Fast hypothetical reasoning using analogy on inference-path networks, in Proc. TAI '93. Boston, MA, USA, 1993, pp. 232–239.

[20] S. Tsuruta and M. Ishizuka, A compiling method for predicate knowledge-base for efficient abductive hypothesis synthesis, J. JSAI, 7 (1992) 130–137 (in Japanese).