A Polynomial-time Predicate-Logic Hypothetical Reasoning by Networked Bubble Propagation Method

Yukio OHSAWA *
Dept. of Systems Eng., Osaka Universitys
Mitsuru ISHIZUKA **
Dept. of Information and Communication Eng., University of Tokyo

Abstract. Hypothetical reasoning is a useful knowledge-processing framework applicable to many problems including system diagnosis, design, etc. However, due to its non-monotonic inference nature, it takes exponential computation-time to find a solution hypotheses-set to prove a given goal. This is also true for cost-based hypothetical reasoning to find an optimal solution with minimal cost. As for the hypothetical reasoning expressed in propositional logic, since it is easily transformed into 0-1 integer programming problem, a polynomial-time method finding a near-optimal solution has been developed so far by employing an approximate solution method of 0-1 integer programming called the Pivot and Complement method. Also, by reforming this method, a network-based inference mechanism called Networked Bubble Propagation (NBP) has been invented by the authors, which allows even faster inference. More importantly, a network-based approach is meaningful, for its potential of being developed extending to a broader framework of knowledge processing. In this paper, we extend the NBP method to dealing with the hypothetical reasoning expressed with predicate logic. By constructing a series of knowledge networks, to which the NBP method is applied, in a stepwise manner according to a top-down control, we avoid the excessive expansion of the network size. As a result, we can achieve a polynomial-time inference for computing a near-optimal solution for the cost-based hypothetical reasoning in predicate-logic knowledge.
Keywords: Reasoning (abduction) , Knowledge Representation, Search

1 Introduction

Hypothetical reasoning [1] is an important knowledge-processing framework for its theoretical basis and its practical applicability to diagnosis, design, etc. Here, knowledge is composed of background knowledge Σ (knowledge given as true in any environment) and hypothetical one H (a set of possible element hypotheses which may contradict with others). Its basic function is to search for a subset h of H, whith which Σ can derive a given goal G. h should be also consistent with Σ, i.e., h should not derive an empty clause when combined with Σ.

* 1-3 Machikaneyama, Toyonaka, Osaka 560, JAPAN email:osawa@sys.es.osaka-u.ac.jp
** 7-3-1 Hongo, Bunkyo-ku, Tokyo 113, JAPAN

Its crucial problem, however, is the slow inference speed. That is, the inconsistency among element hypotheses causes its inference-time to grow exponentially with the number of possible element hypotheses. Its computational complexity has been proven to be NP-complete or NP-hard [2, 3]. To overcome this problem, many symbolic inference approaches have been proposed. Top-down inference starting with the goal, like [4] employing the QSQR approach [5], can focus the search to the useful portion of knowledge for proving the goal. On the other hand, Bottom-up and paralleled inference starting with the hypotheses (ATMS[6], MGTP[7]) can avoid backtrackings. There are also approaches which mixed both merits (inference-path network method[8], or adoptation of the upside-down meta-interpretation[9] into hypothetical reasoning including inconsistency). Despite these efforts, the inference-time still stays intractable, i.e., exponential against the problem size.

This computational intractability, however, can be overcome if we search for a near-optimal solution instead of the exactly optimal one in a cost-based hypothetical reasoning, in which a weight value is assigned to each element hypothesis and an optimal solution hypotheses-set with the minimum sum of the weigh values is searched (its probabilistic semantics is discussed in [10], for example). As for 0-1 integer programming which is also NP-complete, there is an efficient approximate solution method for computing a near-optimal solution, the Pivot and Complement (PC, hereafter) method[11] . It allows a polynomial-time computation for obtaining a near-optimal solution which is quite close to the optimal one (the data presented in [11] in 1980 is still highly ranked among existing methods[12]). This method consists of three phases, i.e., initial, search and improvement phases. In the initial phase, the optimal solution in real domain [0, 1], not in binary domain of 0 or 1, is computed quickly by the simplex method. Then a local search is performed around this initial solution to find a feasible 0-1 solution in the search phase, by pivoting operations which is similar to the vertex-to-vertex tracing in the simplex method. Finally, the value of initially defined objective function is reduced, by a limited times of complement operations (turning over the variables' values between 0 and 1) in the improvement phase. The overall procedure of the PC method takes time in the order of $O(N^4)$, where N is the number of variables. Since the cost-based propositional hypothetical reasoning which is equivalent to a 0-1 integer programming, a solution hypotheses-set with the minimal or the near-minimal cost could be obtained quickly by this PC method.

However, it has been hard to understand the original matrix-based PC method from the viewpoint of knowledge-processing so we invented the Networked Bubble Propagation (NBP) method[13] by reforming the PC method into network-based operations. This method allows us to understand its inference mechanism, by visualizing the operations as network-based semi-symbolic ones. So it serves as a useful bridge connecting mathematical programming and AI. Also, thanks to the focusing of each operation into small subnetworks, it achieved an even faster inference than the original PC method.

In this paper, we take advantage of this network-based nature of the NBP method, i.e., extend it to deal with the cost-based hypothetical reasoning not only in propositional logic but also in function-free predicate Horn clauses. For this purpose, it is straightforward to unfold the predicate clauses into propositional ones in Herbrand domain to apply existing propositional NBP method. However, enormously large number of clauses are generated that way. In order to avoid this knowledge expansion, we invent a new type of knowledge network, in which the arguments of the predicate literals and the constraints among them given as Horn clauses constitute the nodes of the network. This newtwork is extended in a stepwise top-down manner like the QSQR method and avoids from growing too large. Finally, we will show experimentally and theoretically that this new method called predicate-NBP method, achieves a polynomial-time inference against the size of this extended network.

2 Networked Bubble Propagation (NBP) Method

We first outline the NBP method of propositional logic version. Its inference process goes on by propagating a fixed number of "bubbles" along the arcs of the network, until the network reaches a final state. One bubble-propagation is equivalent to one pivoting, i.e., exchange of a basic variable and a non-basic one in the PC method.

For example, let the background knowledge Σ and the set H of possible hypotheses with weights be Eqs.(1), where 'inc' stands for 'inconsistent', which means in this case that X2 and X5 can not coexist in one environment.
Example knowledge-base 1

Σ (Background Knowledge):
G :- X3. G :- X4. X3 :- X1,X2,X5. X4 :- X5,X6. inc :- X2, X5
G (Goal): G
H (Possible Hypotheses):
X1 (weight=1), X2 (weight=2), X5 (weight=1), X6 (weight=2)

(1)

The solution subset h of H is expected to be an optimal one, i.e., of the minimum sum of weights of its member hypotheses and can derive G, without violating the 'inc' constraint.

In order to solve this problem within a polynomial-time by the PC method, we transform the knowledge-base into a set of inequalities as follows. Let the structural variables be represented by italic characters $Xi (i = 1, 2, ...)$ and Y and take the truth-values of same-named logical variables (atoms) Xi and Y, respectively (We use italic for the names of real-valued variables and sets like Σ, and plain ones for others). We transform a propositional Horn clause Y: -X1, X2 , ... Xn into

$$-\frac{n-1}{n} \leq S_{NBP} \leq 0 \tag{2}$$

where the value of an introduced slack variable S_{NBP} is defined as,

$$S_{NBP} = Y - \frac{X1 + X2 + ...Xn}{n} \tag{3}$$

Also, a set of Horn clauses such as Y:- X1, Y:- X2, ..., Y:- Xn, is transformed as,

$$0 \leq S_{NBP} \leq \frac{n-1}{n} \tag{4}$$

where SNBP is again defined in (3). We first fix the values of goal G as 1, 'inc' as 0. While the structural Xi and Y are allowed to take real values in the range of [0, 1] during the process, they must eventually become 0 or 1 in the final state. By solving the 0-1 integer programming thus formed, we obtain a solution hypotheses-set as the collection of hypotheses with value 1.

The NBP method is a networked version of the PC method. Putting its several useful tricks aside, we show here only a bubble-propagation operation. First of all, an inequality constraint (2) or (4) is translated into a constraint network like Fig.1, which becomes a unit component of a "bubble propagation network (briefly BPN)". In this description, the structural variables corresponding to the logical literals appearing in the knowledge base are depicted as circular (structural) nodes. The slack variable SNBP, is depicted as an angular (slack) node. The value of each node is restricted between its upper and lower bounds; this range is [0, 1] for a circular node, and one defined by inequality (2) or (4) for a slack node. The example knowledge-base 1, corresponding to the commonly used

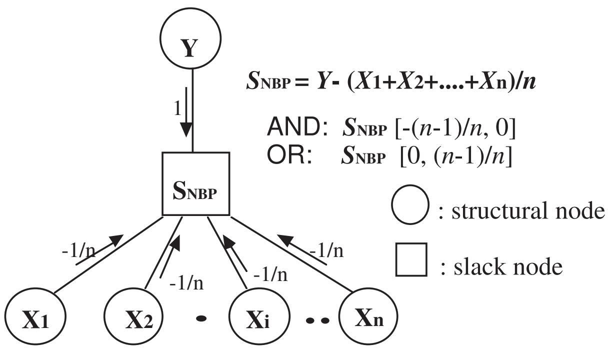

Fig. 1. A unit constraint network of BPN(Each value attached to one arrow is *propagation-rate*, the quantity that a unit variation in a structural node is propagated to node S_{NBP}.)

knowledge network of Fig.2, is transformed into the BPN of Fig.3 by combining the unit components like Fig.1.

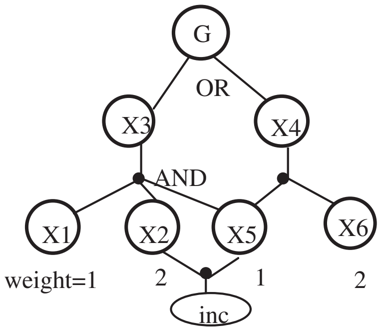

Fig. 2. A networked expression of knowledge-base 1

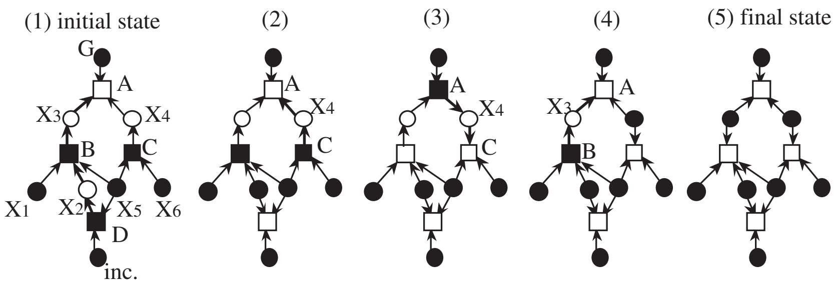

Fig. 3. A series of bubble propagation network (BPN) illustrating an inference process of the NBP method.

We call the black-color of nodes "bubbles", which represent non-basic variables (equal to their upper or lower bounds) in the area of linear programming. On the other hand, white nodes represent basic variables. As well as the pivoting of the simplex method, the searching point moves keeping a constant number of non-basic variables (or nodes), i.e., as many as the number of structural variables or the dimension of the search space.

Note that the $\bullet$ node has its own value as 0 or 1, but the $\circ$ node is determined according to its neighboring $\blacksquare$ nodes. That is, since the $\blacksquare$ node means such an equality as

$$Y - \frac{X1 + X2 + ... + Xn}{n} = const.(upper\ or\ lower\ bound\ of\ S_{NBP}) \quad (5)$$

Xi is determined after all the values $(Y, X1, X2, ...Xi-1, Xi+1, ..., Xn+1)$ are established. Also, the value of the $\square$ node is fixed after all its neighboring nodes. Consequently, the ruling ordering among nodes is given as the arrows of Fig.3, called "ruling arrows".

We show here an example of the inferece process in Fig.3. When a bubble propagates from node D in state (1), it goes down to the basic node which makes the least change in the value of node D after the propagation, although we skip here the details of the numerical calculations involved. This basic node the bubble reaches can exist only in the down stream of the ruling arrows starting from node D. In this case, it moves from node D to node X2 and the BPN changes into state (2). By this restriction of search area in the BPN, the NBP method can infer even faster than the PC method, within a computing time of $0(N^2)$. It reaches the final state (5) when all the bubbles are gathered up to the circular or structural nodes. This method works also for cases where some arcs are left without arrows (see [13] for details).

3 Predicate-NBP Method

3.1 Outline of the Predicate-NBP Method

Predicate literals have arguments which may be constants or variables when we consider function-free literals. In order to apply the NBP method to the predicate-logic hypothetical reasoning, we first consider how to represent the arguments by the BPN nodes. In this paper, we present a novel solution to this problem. Our approach is as follows:

1. Regard a predicate Horn clause as a constraint among variables and translate it into an inequality constraint (see 3.2).
2. In order to realize both high efficiency (also for knowledge including recursive Horn clauses) and a goal-directed inference, we construct the BPN by mixing top-down and bottom-up approaches like the QSQR [5] approach.

Briefly outlining the item 2, here, we call the memory area of the whole knowledge initially given as Σ and H, stored as a set of BPN pieces, "long term memory (LTM)", and the memory area storing the growing BPN "short term memory (STM)". The BPN is constructed in the STM by stepwisely adding pieces of network copied from the LTM, from top(goal) down to the hypotheses level. By selecting these added pieces according to the binding information like QSQR, we can prevent the network from growing too large.

Together with the major merit of item 1, that we do not have to prepare as many clauses as the instances of each head predicate (see 3.2), the network constructed in this manner of item 2 is kept within a reasonable size. Eventually, a polynomial-time inference is achieved thanks to the NBP method, against this tractable size of BPN.

3.2 Translation of Predicate Horn Clause into Inequality Constraints

In this subsection, we describe the details of the item 1 of 3.1. We consider the case where Σ consists of range-restricted function-free Horn clauses, and H

consists of grounded unit clauses. This restriction is loose enough to cover many practical problems.

We first regard a Horn clause as a procedural knowledge. For example, the Horn clause in Fig.4 can be interpreted as "if the first argument(X1)'s value of q1 is equal to the first ones (X2 and X3) of q2 and q3, and also Y1=Y2=Y3 and Z1=Z2, then substitute the second argument(Y)'s value of q1 into Y of p, Z of q2 into Z of p". We divide such a statement into two statements, "if X1, X2,...Xn are equal (more conditions may exist), then switch S can be (instead of is, because of the reason stated later in this subsection) ON "(statement 1) and "if S is ON, then substitute Yin into Yout ; otherwise substitue nothing to Yout "(statement 2), as depicted in Fig.4. We prepare a lng-ary tuple for one

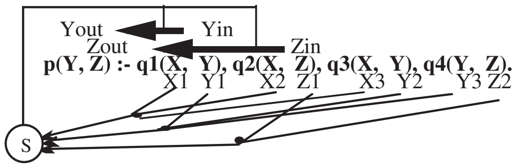

Fig. 4. A procedual interpretation of range-restricted function-free Horn clauses. (S is a switch node to be activated by equality condition.)

argument, where lng is defined as the smallest integer not less than $log_2(m+1)$ where m is the largest size of variables' definition range. A value or an instance of the variable is described in a bit vector of length lng. Suppose the definition range of X for a predicate man(X) be James, Mark, John, m is equal to 3 so that $lng = [log 24] = 2$ and James = 01, Mark = 10, .. John = 11. Thus regarding any instance as a 0-1 integer set, the statements 1 and 2 can be taken in a numerical manner, by interpreting *nothing* in statement 2 as "00...0". Statement 1 can be considered as follows. Let the p-th bits of the bit vectors of X1,X2,...Xn be $T_1^p, T_2^p, ...T_n^p$. Then, equation X1 = X2 = .. Xn is equivalent to $T_1^p = T_2^p = ... = T_n^p$ for any p from 1 to lng. This 0-1 integer equality condition can be described in a constraint form as,

$$1 \leq S + (2 - \frac{1}{n+1})R^p - \frac{\sum_{i=1}^{n} T_i^p}{n+1} \leq 2 \tag{6}$$

where S is the substitution switch variable, where $S = 1$ means it is ON and $S = 0$ OFF. R is a co-variable such that the restriction on R's value to 0 or 1 makes constraint (6) exactly equivalent to statement 1(we omit its proof here, but it can be checked by a values-table over $S, T_1^p, T_2^p, ...T_n^p$, and Rp). Also, statement 2 can be plainly,

$$\frac{Y_{in}^p + S}{2} - \frac{1}{2} \leq Y_{out}^p \leq \frac{Y_{in}^p + S}{2} \tag{7}$$

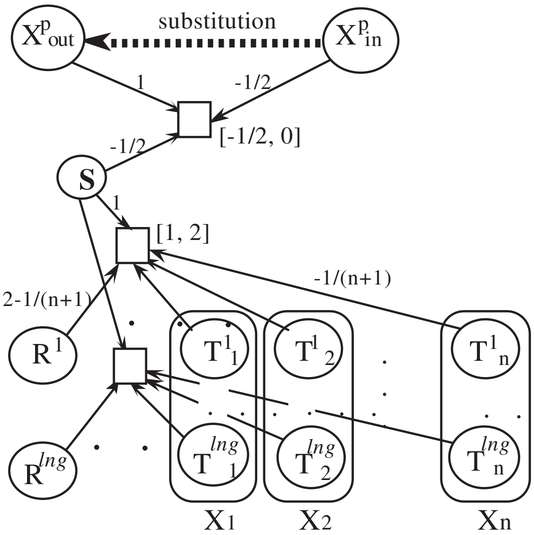

Fig. 5. A networked representation of the substitution switch

Accordingly, statements 1 and 2 can be combined as illustrated in Fig.5. Note that a single switch node S is enough for one Horn clause as in Fig.5, because the "can be" in statement 1 allows the value of S to be 0 or 1 instead of definitely 1 when an equality is detected (do not worry about the stuck-off of switch S because such a state is forbidden by the substituted goal). The knowledge expansion, which occurs if we transform the predicate clauses into propositional ones in Herbrand domain and then to inequality expressions, is avoided by this formalization based on constraints among arguments.

3.3 Dynamic Combination of Substitution Switches

As outlined in 3.1, We employ a top-down BPN extension approach to achieve goal-directed inference. This section illustrates how the STM extension is performed with referring to example 2 below.

Example knowledge-base 2:

Σ (Background Knowledge):
1) g(X, X) :- h1(X).
2) g(X, Y) :- h2(X, Z), g(Z, Y).
3) h1(X) :- h2(X, X).
4) inc :- h1(X), h1(Y), X $\neq$ Y

G (Goal): g(a, b).
H (Possible Hypotheses):

h1(b) (weight=1), h2(a,b) (weight=2), h2(b,b) (weight=3)

Long Term Memory (LTM)
The LTM stores Σ and H, as a collection of pieces of BPN following the formalization in 3.2. As shown in Fig.6 for the case of example 2, the Horn clauses with the same predicate symbol in their heads are gathered into one clustered network. We call this clustered network a BK unit for the background knowledge and a HP unit for the hypotheses. The weight values of the element hypotheses, for example, h2(a, b) and h2(b, b), are assigned to switch nodes S3 and S4 in Fig.6, respectively, since the adoption of these hypotheses are expressed by the ON (or 1) state of these respective switches. For example, the left BK unit in Fig.6 means that g(,) can be derived either from rule 1) or 2) in the background knowledge. Also, since one occurrence of predicate can be instantiated by only one tuple of values, S1 and S2 can not be ON at the same time. Therefore, one must be filtered out by node F1 which makes a constraint,

$$0 \leq F1 \leq 1, where\ F1 = S1 + S2. \tag{8}$$

In Fig.6, F2 in the HP unit for h2(,) is another filtering node.

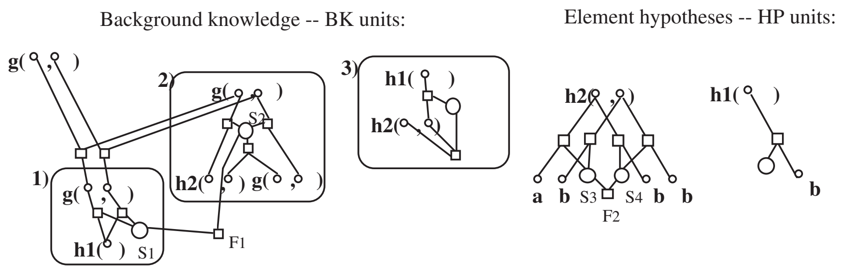

Fig. 6. Fig.6 Long term memory (LTM) consisting of background knowledge and hypotheses.

Under the control of top-down inference, pieces of BPN extracted from LTM is copied and added to the BPN existing in the STM in a stepwise manner. Here, both BK and HP units are used as the components of the STM. Let us show its details next.

Short Term Memory (STM)
Bubble propagations take place in the BPN in the STM, which is constructed by the following procedures (the BPN in the STM is briefly called the STM, hereafter).

Stepwise extension of the STM

1. Initialize the STM as goal g(, ,...), with no arcs yet, and make g(, ,...) be the first member of J, a set of predicates to which heads of units from LTM will be attached in 2.

2. Expand the current STM as follows.

2.1 To each member q of J, copy and attach the BK unit of a corresponding predicate in the head, which fits the binding information of q. Then let the set of new leaves of the STM be dJ. Propagate the binding information of J through arguements down to dJ , and let $J = J - q + dJ$.

2.2 For each member of J, without any BK or HP attached, copy and attach the HP unit of corresponding predicate symbol. Propagate the binding information of this newly attached HP units sideway through arguements to dJ.

2.3 Add the filtering nodes which imply inconsistency relation among hypotheses.

3. Perform the bubble propagation operations on the current STM. If a 0-1 integer solution is obtained, yield the switch nodes with ON state (value 1) as a solution, and stop. Otherwise, go to 2.

The substeps in 2 implies that the units can be attached to the STM only if they do not contradict the binding information propagated from the goal and already-attached hypotheses, which is an idea extracted from the QSQR approach. Fig. 7 illustrates this stepwise extension for the quite simple problem of example 2. Referring to Fig.7, we see below how the predicate-NBP method works.

(0): Initialize STM, so that it includes only G: g(a, b).

(1): BK unit with head g(,) is attached to the STM (step 2.1). Then HP units of h1( ) and h2(,) are attached to the STM (step 2.2). Hypotheses h1( ) is not selected, since the binding information from the goal g(a,b) is not consistent in g(X, X):-h1(X) (because a $\neq$ b). Also, value "a" of the first argument of the goal, propagated down to h2(,), disallows h2(b, b) to be selected. Solution is not obtained yet.

(2): Returning to step 2.1, the STM is again extended. The BK unit with head g(,), HP units of h1(b) and h2(b, b) are attached (h2(a, b) is prohibited by the sideway propagation of value "b" from already-attached h2(a, b)). The inconsistency constraint 4) of example 2 is realized as "inc1", which forces X1=X2 (this covers such an instance as (X1, X2) = (b,ϕ), where $\phi \equiv 00..0$, since S_1' is free to be 0 and makes the value of X1= b meaningless). A 0-1 integer solution is obtained by step 3 at this stage. The nodes with shadow here are ON swich nodes, which means the solution hypotheses and the inference path to prove G. As a result, a set of rules and hypotheses for deriving the goal is found as g(b, b) :- h1(b) and g(a,b) :- h2(a,b), g(b,b) with the solution hypotheses h1(b) and h2(a,b).

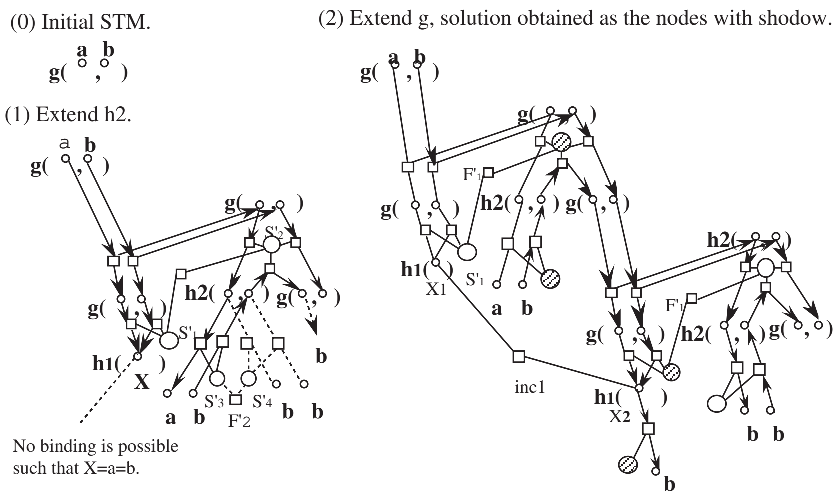

Fig. 7. Stepwise expansion of the predicate BPN

4 Experimental Performance Evaluation

4.1 An Example Problem

This section illustrates experimental results of the predicate-NBP method. The system is implemented in C language on Sun SPARC station 10 using virtual memory of about 200 Mbyte. The problem we consider here is the optimization of a production line. We wish to choose the optimal sequential process to produce "product (Goal)" from "product (Start)", through a sequential adoption of a recursive rule as product(X) :- task(X, Y), product(Y).

Possible transitions among intermediate products to be produced between Start and Goal are given as element hypotheses as high-tool(A, B) whose weight being 1/1000 which means "1/1000 fault rate for producing B from A by high-performance tool," where two classes of tools can be used, i.e., of high- and low-performance. Since a high-performance tool consumes high-power, we also impose a constraint that only one high-performance tool is available for each couple of $(2m-1)-th$ and $(2m)-th$ transitions counting from Start, for any arbitrary integer m. Under this restriction, we wish to minimize the fault rate of the final product, or the sum of all the fault rates of the transitions from the Start to the Goal.

4.2 Performance Estimation

In all the cases, solutions obtained were within the 3 best ones, which are in principle identical to the ones to be obtained if the original matrix-based PC method is applied to the inequality constraints corresponding to the finally constructed STM.

Also, Fig.8 depicts the measured CPU-time of the predicate-NBP method for this problem. We can observe a major tendency in this graph; The increase of the CPU-time is in the polynomial order of $0(|H|^3)$. This time-performance can be explained as follows.

According to [13], the computation-time of the NBP method is nearly $0(N^2)$, where N is the number of nodes in the STM. N is nearly lng (the lengths of bit vectors) times m (variables appearing in all the Horn clauses in the STM), where lng is equal to $log_2(maximum\ number\ of\ possible\ values\ for\ one\ variable)$, which is nearly proportional to $log_2(|H|)$ in this problem. In our current problem, m is in proportion to —h— (h is the solution hypotheses-set), since both are proportional to the number of recursive calls for product( ). As a result, the inference-time of the predicate-NBP method becomes in the order of $(\alpha|H|log_2|H|)^2 = (|h|log_2|H|)^2$, where α is a constant value meaning the number of possible products to be produced in each transition. (For example, in producing a car, we select an engine, tires-set, and so on each from a limited number of alternatives, on a workbench located in a established part of the line. a can be regarded as the average number of those alternatives.) $(\alpha|H|log_2|H|)^2$ is roughly estimated as between $|H|^2$ and $|H|^3$, corresponding to our results.

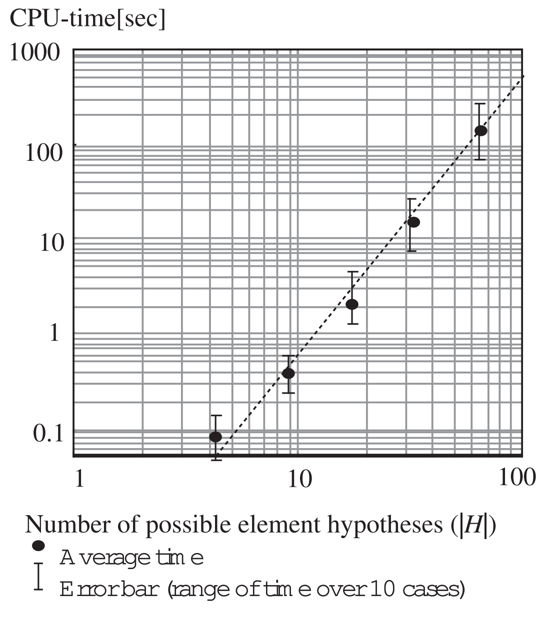

Fig. 8. CPU-time of predicate-NBP method applied to example 3.

5 Conclusion

A polynomial-time approximate solution method of hypothetical reasoning for function-free range-restricted predicate Horn clauses is presented. Theproblem is directly transformed into a network to which the NBP method, a polynomial-time networked inference method to compute a near-optimal solution hypotheses-set, is applied without being unfolded into Herbrand base. Although it has been difficult so far to apply such an operations-research-based approach to predicate-logic, our constraint-based method has overcome this problem and thus avoided impractical explosion of both knowldge-size and inference-time.

Another distinctive feature of our method is its top-down stepwise extension of the bubble propagation network (BPN). By this idea, the network associated with a given goal is trimmed into a goal-directed one. This extension can be guaranteed to be stopped by detecting the halting condition of QSQR even in the case where no solution is available, which will be described in another paper.

References

1. Poole,D., Aleliunas,R. and Goebel,R.: Theorist : A Logical Reasoning System for Defaults and Diagnosis, in the Knowledge Frontier : Essays in the Knowledge Representation (Cercone, N. J. and Macalla, G., eds.) Springer-Verlag, N.Y., 1987.
2. Kautz, H.A and Selman, B.:Hard Problems for Simple Default Logics, Proc. 1st Int'l Conf. on Principles of Knowldge Representation and Reasoning, 189-197, 1989.
3. Bylander,T., Allemang,D., et al.: The Computational Complexity of Abduction, Artif. Intell., Vol.49, 25-60, 1991.
4. Kondo,A. Makino,T. and Ishizuka,M.: Efficient Hypothetical Reasoning System for Predicate-logic Knowledge Base, Knowledge Based Systems, Vol.6, 87-94, 1993.
5. Vieille,L.: Recursive Axioms in Deductive Databases: The Query/Subquery Approach, Proc. Expert Database Systems, 253-267, 1987.
6. de Kleer,J.: An Assumption-based TMS, Artif. Intell, Vol.28, 127-162, 1986.
7. Inoue,K., Ohta,Y. and Hasegawa,R.: Hypothetical Reasoning Systems on the MGTP, ICOT Technical Report, TR-763, 1992.
8. Ishizuka,M. and Ito,F.: Fast Hypothetical Reasoning System using Inference-Path Network, Proc. IEEE Int'l Conf. Tools for AI '91, 352-360, 1991.
9. Bry, F.: Query evalution in recursive databases, bottom-up and top-down reconciled, Data and Knowledge Engineering, Vol.5, 289-312, 1990.
10. Charniak,E. and Shimony,S.E.: Probabilistic Semantics for Cost Based Abduction, Proc. AAAI'90, 106-111, 1990.
11. Balas,E. and Martin,C.: Pivot and Complement – A Heuristic for 0-1 Programming, Management Science, Vol.26, 86-96, 1980.
12. Ishizuka,M. and Okamoto,T.: A Polynomial-time Hypothetical Reasoning employing an Approximate Method of 0-1 Integer Programming for Computing Near-optimal Solution, Proc. 10th Canadian Conf. AI, 179-186, 1994.
13. Ohsawa,Y., Ishizuka,M.: Networked Bubble Propagation Method as a Polynomial-time Hypothetical Reasoning for Computing Quasi-optimal Solution, Proc. IEEE Int'l Conf. Tools with AI (TAI'93), 184-187, 1993.

This article was processed using the LaTeX macro package with LLNCS style