ELSEVIER

Artificial Intelligence 91 (1997) 131–154

Artificial Intelligence

Networked bubble propagation: a polynomial-time hypothetical reasoning method for computing near-optimal solutions

Yukio Ohsawa [a,*], Mitsuru Ishizuka [b,1]

[a] *Department of Systems Engineering, Osaka University, 1-3 Machikaneyama, Toyonaka City, Osaka 560, Japan*

[b] *Department of Information and Communication Engineering, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan*

Received March 1996; revised November 1996

Abstract

Hypothetical reasoning (abduction) is an important knowledge processing framework because of its theoretical basis and its usefulness for solving practical problems including diagnosis, design, etc. In many cases, the most probable hypotheses set for diagnosis or the least expensive one for design is desirable. Cost-based abduction, where a numerical weight is assigned to each hypothesis and an optimal solution hypotheses set with minimal sum of element hypotheses' weights is searched, deals with such problems. However, slow inference speed is its crucial problem: cost-based abduction is NP-complete. In order to achieve a tractable inference of cost-based abduction, we aim at obtaining a nearly, rather than exactly, optimal solution. For this approach, an approximate solution method exploited in mathematical programming is quite beneficial. On the other hand, from the standpoint of knowledge processing, it is also important to realize inference on a network which reflects knowledge structure. Knowledge structure is a fruitful information for an efficient inference. In this paper, we propose an inference method which works on a knowledge network, based on a mechanism similar to the pivot and complement method, an efficient approximate 0–1 integer programming method to find a near-optimal solution within a polynomial time of $O(N^4)$, where N is the number of variables or hypotheses. We reformalize this method by a new type of network on which inference is executed by propagating *bubbles*. This method achieves an inference time of $O(N^2)$ by executing each bubble propagation within a small sub-network, i.e., by taking advantage of the knowledge structure. © 1997 Elsevier Science B.V.

Keywords: Hypothetical reasoning; Knowledge network; Approximate solution method; Polynomial-time inference

* Corresponding author. E-mail: osawa@sys.es.osaka-u.ac.jp.

[1] E-mail: ishizuka@miv.t.u-tokyo.ac.jp.

PII S0004-3702(96)00061-6

1. Introduction

Hypothetical reasoning (abduction) is a useful knowledge system framework applicable to many practical problems such as diagnosis, design, etc. Its logic-based framework is presented in [21,22], where knowledge is divided into two components, i.e., background knowledge Σ and hypothetical knowledge H. Σ is composed of inference rules and facts which are always true, while H consists of hypotheses which may be true or false and may contradict with others. If Σ is not sufficient for deriving a given goal G, i.e., what is observed and wished to be explained, hypothetical reasoning searches for a hypotheses set $h \subseteq H$, which is consistent with Σ and supports G if combined with Σ. That is, the function of hypothetical reasoning is to find a hypotheses set h that satisfies the following constraints.

$$h \subseteq H, \tag{1}$$

$$\Sigma \cup h \vdash G, \tag{2}$$

$$\Sigma \cup h \nvdash \square \text{ (empty clause)}. \tag{3}$$

We call h a solution hypotheses set or simply a *solution set*. As an example, by associating each hypothesis in H with a possible state of a component in one system and G with a set of observed symptoms, by hypothetical reasoning we can perform a model-based diagnosis for finding a set of fault hypotheses (h) consistent with the observations. Also, by associating each hypothesis with a possible component or a connection (between components) and G with a given system specification, we can perform a model-based designing [17].

In many cases, we wish to obtain a hypotheses set which is the best explanation of the goal, i.e., the most likely diagnosis for a system fault, or the least expensive solution for a design. Cost-based abduction deals with this type of problem solving. Here, a numerical weight is assigned to each hypothesis in H and an optimal solution hypotheses set with the minimal *cost* is searched, where cost is sum of the weights of element hypotheses in h. According to [4], this minimal cost hypotheses set is of the greatest posterior probability if each weight is given as $-\log(p)$, where p is the probability of the corresponding element hypothesis. This means that the most likely solution for diagnosis can be obtained by cost-based abduction. Also, for designing a system, the lowest cost product is available by assigning the cost of each component as the weight of each possible hypothesis.

However, a crucial problem of hypothetical reasoning including cost-based abduction is the slow inference speed. That is, consistency checking among possible hypotheses makes the computation time nonmonotonic and causes the inference time to grow exponentially with the number of possible hypotheses (N). The computational complexity has been proven to be NP-complete [15,27]. To overcome this slow inference speed, many symbolic approaches have been proposed. Top-down inference starting from the goal, like [16] which employs the QSQR approach [28], can focus the search to the useful portion of knowledge for proving the goal. On the other hand, bottom-up and parallel inference starting from the hypotheses, as seen in ATMS [6] or MGTP [11],

$$var(X) = \sum_{i=1}^{rank} w(X, S_i) var(S_i), \tag{10}$$

$$\begin{pmatrix} w(T_1, S_1) & w(T_1, S_2) & \cdots & w(T_1, S_{rank}) \\ \vdots & \vdots & & \vdots \\ w(T_{rank}, S_1) & w(T_{rank}, S_2) & \cdots & w(T_{rank}, S_{rank}) \end{pmatrix} \cdot \begin{pmatrix} var(S_1) \\ var(S_2) \\ \vdots \\ var(S_{rank}) \end{pmatrix}$$

$$= \begin{pmatrix} var(T_1) \\ 0 \\ \vdots \end{pmatrix}. \tag{11}$$

The first row of Eq. (11) is the variation in the value of the source node, and the second to the (*rank*)th rows mean that other ■ nodes stays non-basic, i.e., stays with their boundary values (this matrix computation is possible because *rank* is equal to *rank'* after Procedure 3 (see Appendix B), so that there are *rank* − 1 zeroes in the right-hand side of Eq. (11)).

For the example in Fig. 6, $var(S_1) : var(S_2) = 1 : 1$, because the second row of Eq. (11) is $var(S_1) - var(S_2) = 0$. Then the variation ratios among all the nodes are computed by Eq. (10) as

$$var(T_1) : var(S_1) : var(S_2) : var(P) : var(Q) : var(R)$$
$$= -\tfrac{1}{2} : 1 : 1 : 1 : 1 : 1. \tag{12}$$

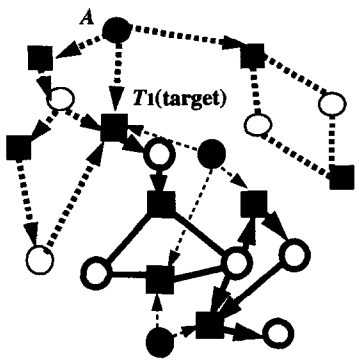
5.1.3. The calculation of relative depths

The *relative depth*, denoted by $depth(X)$, indicates how much the source node's value varies if the value of node X changes to its upper or lower bound. Whether X decreases or increases is determined according to whether the source value is in the lower or upper bound, since the search point should be kept in the feasible region. We select the basic node X with the smallest value of $|depth(X)|$ as the destination node. This corresponds to the minimum shift of the search point from 1 not to 2′ but to 2 in Fig. 1. If $var(T_2) \neq 0$, this $depth(X)$ is defined by Eqs. (13)–(15), where $up(X)$ and $lw(X)$ denote the upper and the lower bounds of node X, respectively.

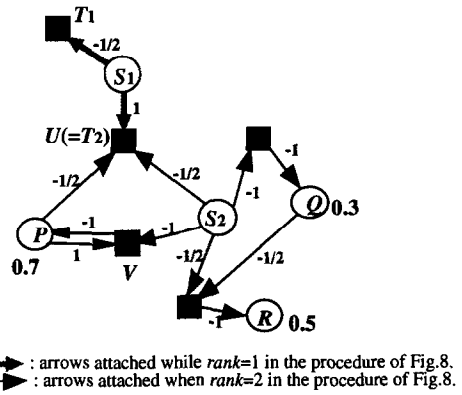
$$\text{if } \left(\frac{up(T_1) + lw(T_1)}{2} - T_1\right) \frac{var(X)}{var(T_1)} > 0 \text{ then } depth(X) = \frac{up(X) - X}{var(X)}, \tag{13}$$

$$\text{if } \left(\frac{up(T_1) + lw(T_1)}{2} - T_1\right) \frac{var(X)}{var(T_1)} < 0 \text{ then } depth(X) = \frac{lw(X) - X}{var(X)}, \tag{14}$$

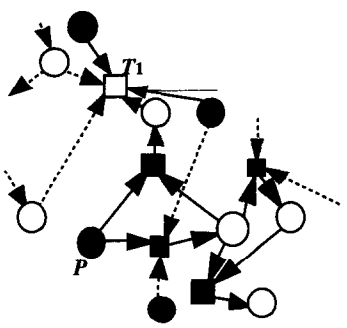
$$\text{if } \left(\frac{up(T_1) + lw(T_1)}{2} - T_1\right) \frac{var(X)}{var(T_1)} = 0 \text{ then } depth(X) = +\infty. \tag{15}$$



(1) Attachment of ruling arrows before procedure 3.



(2) The extracted propagation path with respect to T_1 from the network of (1).



(3) The bubble propagated from T_1 to P.

Fig. 6. An example of computing influence degrees.

The inequalities in Eqs. (13) and (14) indicate the conditions for increasing and decreasing the value of X, respectively. Eq. (15) means that a basic node X cannot be the destination node if $var(X)$ is 0 (the first bracket cannot be 0, because $T_1 = up(T_1)$ or $lw(T_1)$ but $up(T_1) \neq lw(T_1)$, since we do not count the nodes of goal or *inc* whose upper and lower bounds are equal, as T_1). Also, if $var(T_1) = 0$, any ○ node is allowed to be selected since the source node's bounds will never be violated.

In Fig. 6, consider the value change of source node T_1 supposing that the bubble at T_1 propagates to node P. Assume the values before the bubble propagation:

$$(T_1, S_1, S_2, P, Q, R) = (0, 0.1, 0.7, 0.7, 0.3, 0.5).$$

Since $T_1 = up(T_1)$ at this time and $var(P)/var(T_1) < 0$ according to Eq. (12), $depth(P)$ is computed as 0.3 from Eq. (13). Similarly, the relative depths of nodes are calculated as $(1, 0.9, 0.3, 0.3, 0.7, 0.5)$. Since $|depth(P)|$ is the smallest, the bubble at T_1 propagates to P, with an increase of 0.3 in the value of P. Thus the variance of the variables is computed as

can avoid backtracking. There are also approaches in which both of the above merits are combined, such as the inference path network method [12], or a method [20] based on an extension of the upside-down meta-interpretation [3]. Despite these efforts, the inference time of hypothetical reasoning remains intractable, i.e., exponential against N.

This computational intractability, however, can be overcome if we search for a near-optimal solution rather than the exactly optimal one in cost-based abduction. An efficient method for computing near-optimal solutions in 0–1 integer programming (also NP-complete) has been developed in operations research [2]. This method, pivot and complement (PC, hereafter), finds a near-optimal solution which is quite close to the optimal solution in a polynomial-time method. The overall procedure of the PC method takes time $O(N^4)$, where N is the number of variables. Hence the key insight in [13]: a cost-based abduction problem represented in propositional logic can be translated into an equivalent 0–1 integer programming problem. Then a solution set with the minimal or a near-minimal cost can be obtained quickly by the PC method (this performance is one of the best among existing methods, both in solution quality and the computation time [13]).

However, the numerical calculations of the PC method is a sort of black box from the standpoint of knowledge processing; it is difficult to improve the procedure by considering particular knowledge structures. Moreover, by understanding the meaning of effective mathematical programming heuristics from a viewpoint of knowledge processing, we obtain a bridge to extend them to a richer knowledge expression, such as predicate logic [19]. Thus, we wish to develop an inference method which works on a knowledge network representing the knowledge structure using a mechanism similar to fast approximate 0–1 integer programming. Also, apart from constructing this bridge from operations research to knowledge processing, using such a network-based mechanism, we can achieve inference even faster than the original PC method. That is, by executing each operation within a focused sub-network rather than the overall network, the inference time can be reduced.

In this paper, we translate the PC mechanism into a network-based process. Very briefly, propositions are represented as nodes in the network and truth values of propositions are defined as real numbers in the range of $[0, 1]$. We introduce *bubble propagation* as a method for propagating truth values along the arcs in the network.

The inference time is reduced to $O(N^2)$ or less, which is faster than the $O(N^4)$ performance of [13] (in [13], the matrix-based calculations of the original PC method proposed in [2] are employed). Also, the solutions are as good as [13], i.e., almost exactly optimal in practical cases.

The remainder of the present paper is as follows. In Section 2, we show how a cost-based abduction problem is translated into an integer programming one. Then, in Section 3, the PC method is outlined. Section 4 explains how the PC method is networked, i.e., how a constraint network called a *bubble propagation network* is constructed from the knowledge base. In Section 5, we illustrate how the inference runs on this network by our new *networked bubble propagation* method. The time performance and the solution quality are estimated both theoretically (Section 6) and empirically (Section 7), and some conclusional remarks are given in Section 8.

2. Translation of cost-based abduction into 0–1 integer programming

We consider cost-based abduction of propositional Horn clauses in this paper. Each possible hypothesis is represented as a single atom with a weight value. For example, let the background knowledge Σ and the set of possible weighted hypotheses H represent the following knowledge base (Knowledge Base 1), where G is the goal and *inc* stands for "inconsistent". The clause with *inc* in the head means in this case that X_2 and X_5 cannot coexist in one environment. The values of G and *inc* are anchored to 1 (both the upper and the lower bounds of the truth value of G are defined to be 1) and 0, respectively.

Knowledge Base 1.

$$\Sigma: \quad G:-X_3, \quad G:-X_4, \quad X_3:-X_1,X_2,X_5,$$
$$X_4:-X_5,X_6, \quad inc:-X_2,X_5.$$
$$H: \quad X_1,X_2,X_5,X_6.$$

The solution set h ($\subseteq H$) is expected to be optimal, i.e., the sum of weights of member hypotheses (given in Eq. (4)) of h is wanted to be minimal, and must also support G without violating the constraints of *inc*.

$$\begin{aligned} &weight(X_1) = 1, \quad &&weight(X_2) = 2, \\ &weight(X_5) = 1, \quad &&weight(X_6) = 2. \end{aligned} \tag{4}$$

In order to make an integer programming method applicable, we translate the knowledge base into inequality-constraints. In the first place, let us define *structural variables* X_i ($i = 1,2,\ldots$) and Y, as variables which take 1/0 values corresponding to true/false of same-named atoms (propositions) X_i and Y. Their values are restricted to $[0,1]$. We transform a propositional Horn clause $Y:-X_1,X_2,\ldots,X_n$ (AND-rule hereafter) into Eq. (5). Also, a set of Horn clauses sharing the same head (Y, i.e., the conclusional atom) like $Y:-X_1, Y:-X_2,\ldots,Y:-X_n$, (OR-rule hereafter) is transformed into Eq. (6).

$$S \geqslant 1-n, \qquad T_i \leqslant 0. \tag{5}$$

$$S \leqslant 0, \qquad T_i \geqslant 0. \tag{6}$$

Here, the *slack variables* are variables which take the values of the slacks of inequality constraints, like S and T_i ($i = 1,2,\ldots,n$) which are defined as

$$S = Y - \left(\sum_{i=1}^{n} X_i\right), \qquad T_i = Y - X_i. \tag{7}$$

We set the value of goal G as 1, *inc* as 0. That is, inconsistency among hypotheses like $inc:-X_2,X_5$ are taken for Horn clauses with false heads (the empty clause). Under thus defined constraints, a feasible *lattice point* (assignment of all the truth values of

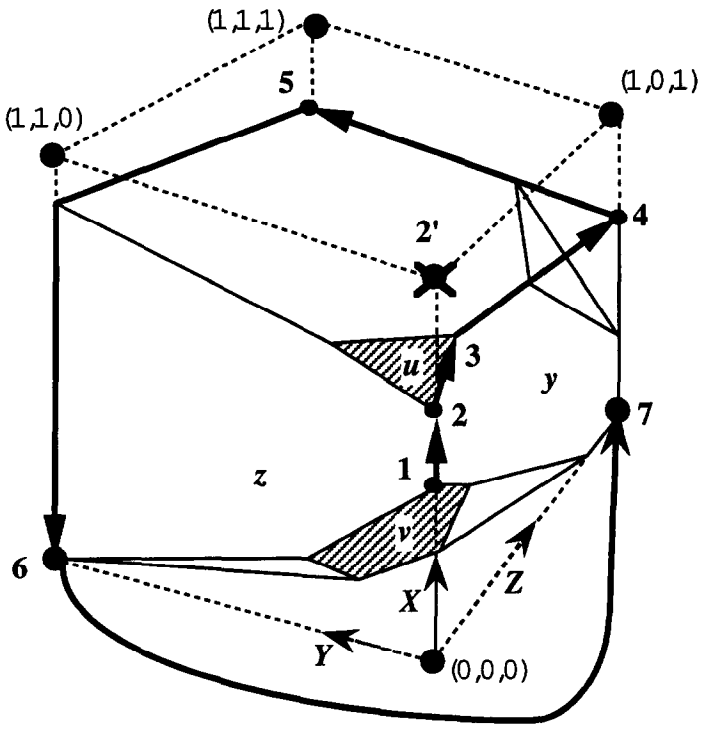


Fig. 1. A visual illustration of the PC for three variables, X, Y and Z (the vertices with larger black dot marks are lattice points).

atoms to an integer value of 1 or 0) with the optimal (minimal) *cost* (the sum of weights of element hypotheses in the solution set h) is searched. The set of element hypotheses with truth values of 1 is interpreted as the solution hypotheses set h. Thus cost-based abduction is translated into 0–1 integer programming.

3. Outline of the pivot and complement (PC)

Our approach is based on a networked reformalization of the PC method [2], which is an efficient approximate solution method for 0–1 integer programming. Since the PC method is an approximate method, its output solution may not be exactly optimal (i.e., of minimum cost). However, solutions obtained are quite near the optimal ones; the detailed results in [2] show that for nearly 1000 involved variables, the difference between the optimal and the near-optimal solutions are within 10^{-2} ordered magnitude against the optimal value of the object function. This performance is highly ranked among existing 0–1 integer programming methods so far.

Let us outline the PC method, referring to the example illustrated in Fig. 1, where three structural variables X, Y and Z exist. In Fig. 1, the solid line polyhedron is the

feasible region where all the constraints are satisfied, and whose surfaces are the equality bounds of inequality constraints. For example, $2X - Y - 2Z = 1.4$ (the striped surface denoted as u), is the bound for the constraint $U (\equiv 2X - Y - 2Z - 1.4) \leqslant 0$. The task of 0–1 integer programming is to search for a lattice point in this feasible region, where a defined objective function is minimized.

A search point is an intersection of as many surfaces as the dimension of the search space, which corresponds to the number of structural variables (surfaces v, y and z intersect to form vertex 1 in Fig. 1). Variables for surfaces that touch a search point are called *non-basic* variables (for vertex 1, non-basic variables are $V (\equiv X + Y + Z - 0.5)$, Y and Z which correspond to surfaces v: $X + Y + Z = 0.5$, y: $Y = 0$, and z: $Z = 0$, respectively). Variables other than non-basic variables are called *basic* variables.

The PC method is a local search method consisting of two phases. The search begins at the point of exact optimal cost, which can be found quickly by a linear programming method [5, 14] within polynomial time (the method of [5] is not strictly polynomial, but runs in polynomial time for most cases) (see also [24]). The initial point is most likely a non-lattice point (vertex 1 in Fig. 1), so the search point moves from vertex to vertex of the polyhedron, until a lattice point (vertex 6 in Fig. 1) is reached (the local *search phase*). In the next phase, the *improvement phase*, an attempt is made to find a better solution. That is, the value of the objective function is reduced by *complementing* the current lattice point solution, i.e., by flipping the values of structural variables from 0 to 1 and vise versa (moving from vertex 6 to vertex 7 in Fig. 1).

Moving the search point from vertex to vertex in the search phase is executed by *pivoting*, exchanging a non-basic with a basic variable which changes the value of the non-basic one, in the direction to stay in the feasible region. Here, the basic one should be selected as to change the value of the non-basic one by the least variation. For example, when the search point moves from vertex 1 $(0.5, 0, 0)$ in Fig. 1, it moves in the direction of $+X$, to prevent violation of the constraint $V \geqslant 0$. In this case, the search point moves to vertex 2 $(0.7, 0, 0)$, not up to vertex $2'$ $(1, 0, 0)$, because vertex $2'$ violates the constraint $U \leqslant 0$ which is bounded by surface u. Thus, the non-basic variables change from V, Y, Z to U, Y, Z, i.e., V which was non-basic at vertex 1 becomes basic at vertex 2 and U changes from basic to non-basic.

In the search phase, it may occur that the next vertex cannot be found nearer to a lattice point than the current position. In the PC procedure, such a standstill is broken through and the search continues, chiefly by violating some constraint once (from vertex 3 to 4 in Fig. 1) and then going back into the feasible region by complementing one or two 0–1 integer values (4 and 5 in Fig. 1). Complementing is also executed in the improvement phase to reduce the cost value (7 in Fig. 1). Pivoting takes most of the total time taken for this search phase.

For easy comprehension of later sections, we confess here that our main strategy is to replace the time-consuming pivoting with the propagation of *bubbles*, whose function is similar to pivoting but which runs faster. That is, the pairs of basic and non-basic variables to be exchanged are selected faster by the propagation of bubbles than by pivoting in the matrix-based PC method, due to the restriction of pivoting operations to a single propagation path.

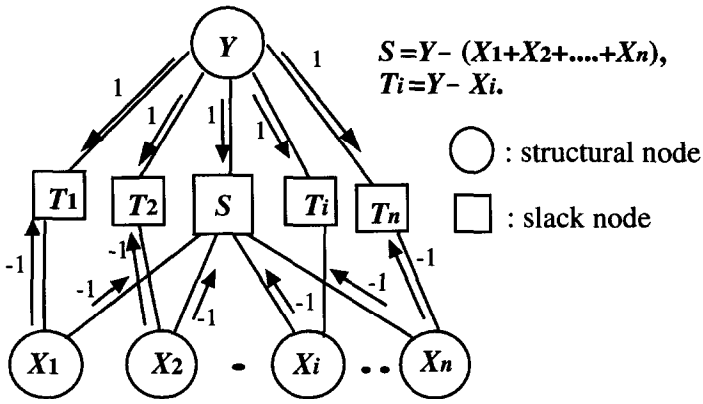


Fig. 2. A unit constraint network of BPN.

4. Networked bubble propagation (NBP) method

In this section, we translate the search phase into a networked mechanism, which we call *networked bubble propagation* (NBP hereafter). We focus on how pivoting is realized by network-based operations.

4.1. Construction of a bubble propagation network (BPN)

Our NBP method works on a *bubble propagation network* (BPN), which is constructed by connecting unit networks each corresponding to one AND- or OR-rule; the BPN thus constructed corresponds to the overall background knowledge. The possible hypotheses are attached as the leaves. We first show a unit constraint network of BPN, and then an overall BPN in this subsection.

Eqs. (5) and (6) can be networked as shown in Fig. 2, where circular nodes are called *structural nodes* associated with structural variables, i.e., atoms. Square nodes which are called *slack nodes* like S or T_i ($i = 1,2,\ldots,n$) are associated with slack variables. The values of nodes are constrained by their upper and lower bounds, i.e., the slack nodes are constrained by Eqs. (5) or (6) and the structural nodes by $[0,1]$. Arcs connect structural and slack nodes and have *propagation rates*. A propagation rate $p(B,A)$ from a structural node A to a slack node B is the variation in the value of B with respect to a unit value variation in node A via the arrow which directly points B from A. That is, the propagation rates from structural to slack nodes are determined on the coefficients of Eq. (7), e.g.,

$$\begin{aligned} p(S,X_i) &= -1, \qquad p(S,Y) = 1, \\ p(T_i,X_i) &= -1, \qquad p(T_i,Y) = 1, \end{aligned} \tag{8}$$

for $i = 1,\ldots,n$. The propagation rate $p(A,B)$ in the opposite direction (from B to A) is defined as $-1/p(B,A)$ (see Appendix A).

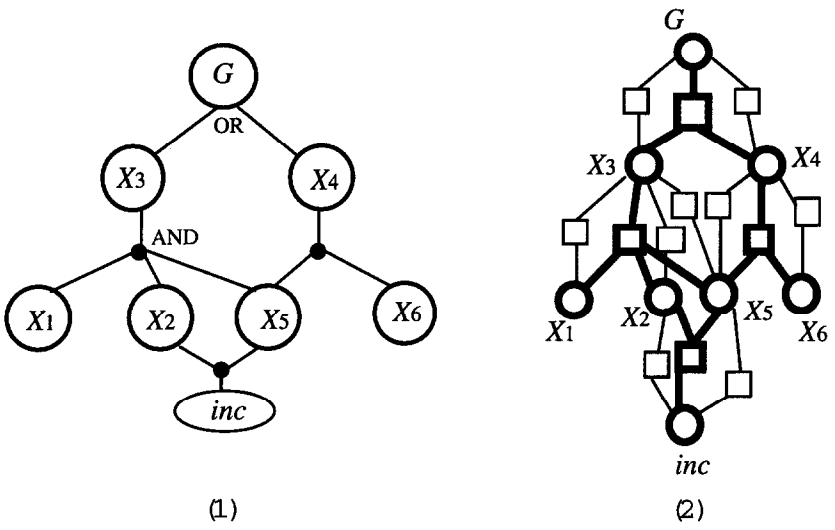


Fig. 3. Networked representations of Knowledge Base 1. (1) Traditional network. (2) Bubble propagation network (BPN).

Let us show how a BPN is constructed, referring to Knowledge Base 1. Fig. 3(1) shows a traditional type of knowledge network used in previous inference methods such as [12], for Knowledge Base 1. The same knowledge base is represented by a BPN in Fig. 3(2) by combining unit BPNs like Fig. 2.

As illustrated in Fig. 4 (this figure is referred to in Section 5.3), we use the node representation ○ for a basic structural, ● for a non-basic structural, □ for a basic slack and ■ for a non-basic slack node. We call the black color of a node, which means the node is in its non-basic state, a *bubble*. The position of a search point is determined by a constant number of non-basic variables (as many as structural variables). A state of a BPN is determined by the location of bubbles in our NBP method. In the example in Fig. 4, there are eight structural nodes which correspond to an eight-dimensional space in the PC method. Thus a search point is determined by eight intersecting surfaces. This means that there are eight non-basic variables.

The initial state of the BPN determined by linear programming is shown in Fig. 4(1). During the search phase, the BPN transits from the initial to the final state by propagating bubbles, like the one propagated from node A to X_1 in changing from state 1 to 2 in Fig. 4. The propagation of a bubble is equivalent to a pivot in the PC method, as stated later in Section 5. The search phase ends when all the bubbles gather at the structural nodes.

4.2. Ruling arrows and propagation path

The most important key to the fast inference of the NBP method is that bubbles move only in the direction of *ruling arrows*, which considerably reduces the time for bubble

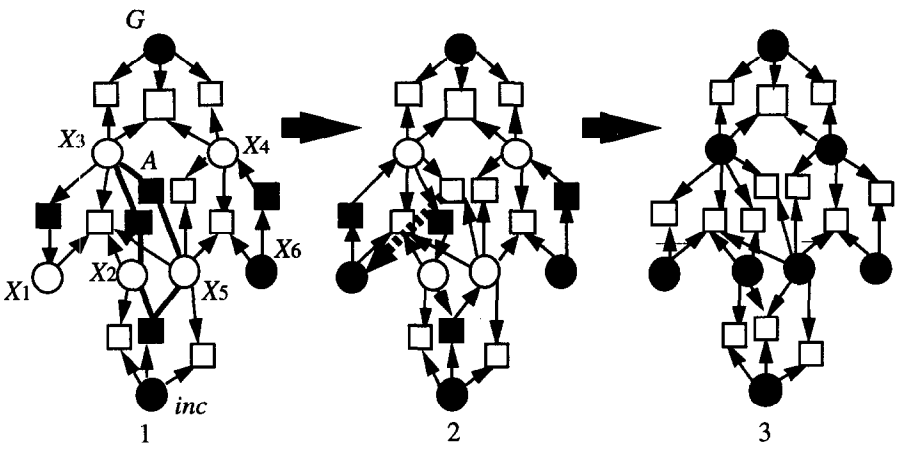


Fig. 4. Transitions of BPN states during the inference.

propagation. That is, the destination node to which a bubble moves (equivalent to a basic variable with which a non-basic variable is exchanged in a pivot) is restricted within a small sub-network (*propagation path*) determined by ruling arrows.

A ruling arrow indicates that its initial node rules the value of the terminal node. More specifically, a ● node determines its own value as 0 or 1 and rules the values of neighboring nodes. On the other hand, the value of a ○ node is determined according to its neighboring ■ nodes. For example, in Eq. (7), X_i is determined after all the values $(Y, X_1, X_2, \ldots, X_{i-1}, X_{i+1}, \ldots, X_{n+1})$ are determined, if S is ■ (i.e., $S = 0$). Also, the value of the □ node is fixed after all of its neighboring nodes are determined. This ordering of ruling among nodes is given by the following procedure (Procedure 1), and depicted by ruling arrows in the network. Fig. 5 illustrates how ruling arrows are attached to the BPN in Fig. 4(1) by Procedure 1.

Procedure 1 (*Attaching ruling arrows*).

1. Attach out-going arrows from ● nodes to all their adjacent nodes.
2. From each ■ node to which in-coming arrows are attached from all the adjacent nodes except one, attach an out-going arrow to this excepted node.
3. From each ○ node to which an arrow is in-coming from an adjacent node, attach out-going arrows to all other adjacent nodes.
4. Attach in-coming arrows to □ nodes from all their adjacent nodes.
5. To each ○ node from which out-going arrows are attached to all the adjacent nodes except one, attach an in-coming arrow to this excepted node.
6. To each ■ node from which an arrow is out-coming to an adjacent node, attach in-coming arrows from all other adjacent nodes.

In Fig. 4(1), ruling arrows are attached to all edges except the six bold lined edges. Since there exists no directed path between X_2 and X_5, neither of these two nodes rules the other's value.

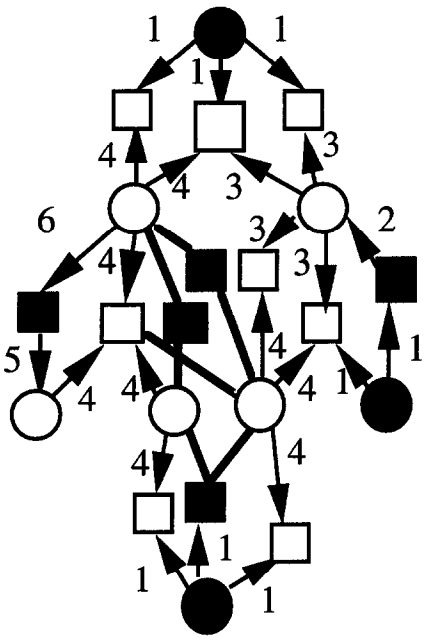


Fig. 5. Attaching ruling arrows to the BPN in Fig. 4(1) (the numbers indicate the step in Procedure 1, by which arrows are attached).

A propagation path is a sequence of edges starting from a *source node*, the non-basic node from which a bubble propagates following ruling arrows (edges without arrows may also be traced). Specifically, a propagation path is given by Procedure 2. A bubble propagates from a source node only to a basic node in its propagation path, because the ruling relation among nodes must be consistent. For example, if the bubble at node A propagates to node *inc* in Fig. 4(1), then *inc* would have two bubbles. This is inconsistent or redundant. Also, values of the eight structural nodes will not be determined uniquely because the number of bubbles would be reduced to only seven. On the other hand, the bubble propagation from node A to X_2 or X_1 is free from this problem since X_1 and X_2 are both in the propagation path of node A. Each step of bubble propagation from a non-basic node (● or ■) is thus focused into a propagation path.

Procedure 2 (*Determining a propagation path*).

1. *Propagation path* = *source node.*
2. Select one node in propagation path not yet *traced*, and call it Z. If there is no such node, stop.
3. Add E: {edges touching Z, except those with ruling arrows received by Z} and V: {nodes on the edges in E} to propagation path. Set Z as *traced*, and go to 2.

5. Mechanism of the NBP method

The mechanism of the NBP method is explained in this section. We explain the bubble propagation process in Section 5.1, and supplement our method with some other minor improvements in Section 5.2. Then in Section 5.3, we illustrate an example of NBP process for Knowledge Base 1.

5.1. Numerical calculation for bubble propagation

A *destination node*, a basic node to which a bubble moves from a source node, should be selected so as to change the value of the source node in the least (see Section 3). To select the destination node, the following calculations are necessary.

(1) *The calculation of influence degrees.* The *influence degree*, defined by $w(X, S_i)$ representing the variation in the value of X with respect to a unit variation in node S_i ($i = 1, 2, \ldots, rank$) (some representative ○ nodes, where *rank* is later determined in Procedure 3), can be computed by summing the products of propagation rates of edges along the path from S_i to X.

(2) *The calculation of variation ratio.* The *variation ratio* is the ratio among $var(S_i)$ ($i = 1, 2, \ldots$), the variations in basic structural nodes S_i ($i = 1, 2, \ldots$) on the propagation path. This variation ratio is computed by solving simultaneous equations whose coefficients are given by influence degrees, under the constraint that the values of ■ nodes other than the source node do not vary by the bubble propagation. The variation ratio among variables other than S_i ($i = 1, 2, \ldots$) in the propagation path is obtained later using $w(X, S_i)$ for $var(X)$.

(3) *The calculation of relative depths.* From the variation ratio, we can tell which node reaches its own lower or upper bound for the minimal simultaneous change in the source, or which basic node is of the minimal *relative depth*.

In the following, we explain some details of items (1)–(3) above.

5.1.1. The calculation of influence degrees

$w(X, S_i)$ is computed by summing all the influences from S_i to X via the paths of arrows (not only ruling arrows but also ones temporarily attached here), where influence is the product of propagation rates of the edges along a path. This computation goes on until these arrows cover the propagation path. More specifically, Procedure 3 is executed. Here, sets I and J are the sets of nodes by which some arrows are received in this procedure, and nodes from which no arrows are spouted yet in this procedure, respectively.

Procedure 3 (*Computing influence degrees* (*the source node is given as* T_1 *and its propagation path* Π *determined by Procedure* 2)).

1. $rank = 1$, $rank' = 1$. Let $S_1 \in \Pi$ be a ○ node adjacent to T_1 and let I be $\{S_1, T_1\}$. Detach all the arrows in Π and let J be the set of all nodes in Π. Regard T_1 as a basic node temporarily until the end of the procedure.
2. $w(S_{rank}, S_{rank}) = 1$, and $w(X, S_{rank}) = 0$ for any node $X \in \Pi$ ($X \neq S_{rank}$).

3. Select a ○ node in $I \cap J$ and call it A. If such a node does not exist, select a ■ node with ngb or $ngb-1$ in-coming arrows and call it B, where ngb is the number of edges touching B. Then remove B from J and go to 5. If there is no such node B, go to 7.
4. Attach out-going arrows to all the non-arrowed edges touching A. For each ■ node (call this node C) at the ends of these arrows,

$$w(C,S_i) = w(C,S_i) + p(C,A)w(A,S_i) \quad (i = 1,2,\ldots,rank),$$

and add C to I. Go to 3.
5. If $(ngb-1)$ arrows are in-coming to B, attach an arrow to the remaining one edge touching B. Call this arrow's end ○ node D. Then add D to I, and

$$w(D,S_i) = w(D,S_i) + p(D,B)w(B,S_i).$$

6. If ngb arrows are in-coming to B, $rank' = rank' + 1$ and $T_{rank'} = B$. Go to 3.
7. If there is an adjacent ○ node in I to a node in I, call it node E. Then $rank = rank + 1$, $S_{rank} = E$, add E to I and go to 2. Otherwise, stop.

An example of Procedure 3 is illustrated in Fig. 6. The bold solid lines in Fig. 6(1) form the propagation path of source T_1. This path is extracted and magnified in Fig. 6(2). Let us assume the propagation rate values of each edge as depicted in the figure.

While $rank = 1$, beginning with $w(S_1,S_1) = 1$, we obtain $w(T_1,S_1) = -\frac{1}{2}$ and $w(U,S_1) = 1$, from $p(T_1,S_1) = -\frac{1}{2}$ and $p(U,S_1) = 1$. Below, we denote the variation in the value of node X as $var(X)$. Next, while $rank = 2$, node S_2 is selected in step 7 of Procedure 3. We obtain $w(V,S_2) = -1$ from $p(V,S_2) = -1$, and then $w(P,S_2) = 1$ from $p(P,V) = -1$ (see Appendix A for how to determine $p(P,V)$). The sequence of such operations proceeds until $w(U,S_2) = -1$ is obtained and then $T_2 = U$ in step 6. As a result, the following influence degrees are obtained and Procedure 3 stops.

$$\begin{pmatrix} w(T_1,S_1) & w(T_1,S_2) \\ w(T_2,S_1) & w(T_2,S_2) \\ w(P,S_1) & w(P,S_2) \\ w(Q,S_1) & w(Q,S_2) \\ w(R,S_1) & w(R,S_2) \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} & 0 \\ 1 & -1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}. \tag{9}$$

5.1.2. The calculation of variation ratio

The variation ratio is the ratio among the value variations of nodes on the propagation path, each variation denoted as $var(X)$ for a basic node or source node T_1 (T_1 is included because it becomes basic after the bubble propagates from T_1). Since the variation ratio of all the nodes can be computed by Eq. (10) (X is an arbitrary node in the propagation path) from the variation ratio $var(S_1) : var(S_2) : \cdots : var(S_{rank})$, we first obtain $var(S_1) : var(S_2) : \cdots : var(S_{rank})$ by solving Eq. (11) on the influence degrees obtained by Procedure 3.

$$(var(T_1), var(S_1), var(S_2), var(P), var(Q), var(R))$$
$$= (-0.15, +0.3, +0.3, +0.3, +0.3, +0.3).$$

Eventually, after being changed by these variances, the values become

$$(T_1, S_1, S_2, P, Q, R) = (-0.15, 0.4, 1.0, 1.0, 0.6, 0.8).$$

The goal of these calculations (of influence degrees, variation ratios, and relative depths) is to find the destination node to which a bubble propagates without violating the constraints. In fact, the values obtained here satisfy the constraints of their upper and lower bounds. Fig. 6(3) shows the network state after all above operations and re-attaching of ruling arrows to edges in the propagation path (which were detached in step 1 of Procedure 3).

5.2. *Other refinements in the NBP method*

Along with our main strategy of realizing pivoting by propagation of bubbles as described above, some refinements have been incorporated for faster and more accurate inference.

Device 1. We prefer ■ nodes to ● nodes in selecting sources (T_1 in Fig. 6). That is, we execute bubble propagation from ● to ○ only if none from ■ is possible, for the following reasons.

(1) If a ● node becomes the source node, the propagation path is large. For example, if the source is node A in Fig. 6(1), we have to select the destination from the nine basic nodes in the area of all the bold lines including both solid and dotted ones.

(2) Turning a node from ● to ○ and returning it to ● afterwards is a round-about way, which can also be done by complementing its value. In fact, according to [2], pivoting between ■ and ○ is prevalent in the PC method.

Device 2. In the search phase we focus the complement, if any, on the ● nodes which can be reached by tracing edges, not in the direction of ruling arrows starting from (i.e., ● nodes whose propagation path includes) a □ node which is violating its constraint. This is because the violation is due only to such a ● node.

Device 3. For the improvement phase, we employ a bottom-up inference process executable in linear time against the network size N, as shown below.

- *Improvement phase (revised).* Test whether the goal is derived if one hypothesis whose truth value is now 1 is denied. If derived, then actually deny this hypothesis and those atoms which are denied accordingly with this hypothesis in the test. Perform this action iteratively for each true hypothesis.

This procedure can improve the solution to a better one than one to be obtained by the original improvement phase of the PC method, as well as the time reduction from $O(N^3)$ (the PC method tries complementing at most three variables) to $O(N)$. That is,

if there are more than 3 atoms between the goal and a hypothesis, complementing less than 4 atoms set in the PC method may violate the constraint. Our new step improves the *cost* of the solution set in such a case too.

5.3. An example behavior of the NBP method

We see below how the NBP method works for the problem of Knowledge Base 1. It reaches the initial state 1 of Fig. 4 by solving a linear programming problem. The transition from state 1 to the final state 3 is shown in Fig. 4.

- *State* 1 *to* 2: In state 1, the values are

$$(X_1, X_2, X_3, X_4, X_5, X_6) = (0.5, 0.5, 0.5, 1, 0.5, 1).$$

A bubble propagates from the source $T_1 = A$. X_2 becomes the ○ node S_1 in Procedure 3. While *rank* = 1 for this source, influence rates are obtained as

$$w(X_3, X_2) = p(X_3, C)p(C, X_2) = 1,$$
$$w(X_5, X_2) = p(X_5, B)p(B, X_2) = -1,$$
$$\vdots$$

Then

$$var(X_1) : var(X_2) : var(X_5) : var(A) : var(\text{other white nodes} \in \Pi)$$
$$= 1 : 1 : 1 : -2 : -1,$$

where Π is the propagation path of node A (nodes and edges which can be traced without opposing ruling arrows starting from A, so that Π includes the bold lined non-arrowed edges in Fig. 4(1)). $var(X_1)$ is one of the least variation, so that the bubble in A propagates to X_1 and reaches state 2. At this moment, the values become

$$(X_1, X_2, X_3, X_4, X_5, X_6) = (0, 0, 0, 1, 1, 1).$$

- *State* 2 *to* 3: Change ○ nodes which take values of 0 or 1 into ●. Since the solution obtained at this moment is the exactly optimal, i.e., X_5, X_6, the improvement phase does not run and the inference stops here.

6. The complexity of NBP and the knowledge structure

The hypothetical reasoning can be classified into problems of different complexity as reported in [15,27]. One of their important conclusions is that the computational complexity of the inference depends on the knowledge structure. A *singly connected graph* is a graph which includes no closed paths. If a knowledge network is singly connected, no inconsistencies exist among elementary hypotheses. As well as the search time of a constraint satisfaction problem [7,8], the inference time of hypothetical

reasoning is polynomial against the number of possible hypotheses if the knowledge network is singly connected [15].

In this section, we show that the computational complexity depends on the knowledge structure in our NBP. The main point is covered by Theorem 1.

Theorem 1. *If the original knowledge network is singly connected, then all* ○ *nodes take* 0 *or* 1 *as their values.*

This theorem means that if the knowledge network is singly connected, all the structural variables take values of 0 or 1 as the output of the initial linear programming procedure, so that it is needless to execute the search phase (to be accurate, our BPN differs from the *primal* or *dual* constraint network employed in [7,8] in that a singly connected BPN may not be translated into a singly connected primal or dual network, i.e., wider range of problems can be put into singly connected BPNs). Theorem 1 is derived from Lemmas 2 and 3, which are proven in Appendix C.

Lemma 2. *If a ruling arrow is attached to every edge in a BPN, then all the* ○ *nodes take values of* 0 *or* 1.

Lemma 3. *A ruling arrow is attached to every edge in a BPN if the overall knowledge network is singly connected.*

Besides deriving Theorem 1, Lemma 2 means that the NBP method reduces a problem into easier ones, because ruling arrows increases with the number of ● nodes as bubbles propagate from □ to ● nodes. For example in Fig. 4, all the values of ○ nodes are already 0 or 1 in state 2 since all the edges in the BPN are attached ruling arrows. Such a structure dependency of inference time is itself a good feature of our method since it does not take unnecessarily long time regardless of the problem structure. Note that Theorem 1 can be extended to the one in [25], when no inconsistencies exist between hypotheses.

7. Experimental evaluation of the NBP method

7.1. Solution quality for test cases

We have implemented the NBP method on a Sun SPARCstation 10 in C. The maximum main (virtual) memory used for experiments reported here was about 83 Mbytes. We applied the NBP method to hypothetical reasoning problems including from 10 to 300 possible hypotheses. We generated test problems randomly, under the condition that each body of the Horn clauses in the background knowledge were within 5 atoms. An atom appeared at most 10 times in the overall background knowledge. These restrictions were for unity of density—we want to know the dependency of time on the problem size not on other factors.

We evaluated the solutions obtained in each test case by cost. For all the cases except three out of 250 problems, the solutions obtained were among the best three solutions. For these excepted three cases, both the PC method and the NBP method could not recover back to the feasible region after temporarily violating a constraint to break a standstill (like vertex 4 in Fig. 1).

Since the NBP method was invented by reformalizing pivoting operations in the PC method to networked operations, the PC and the NBP methods obtained exactly same solutions by the search phase for all the problems. However, due to Device 3 in Section 5.2, the NBP method successfully obtained the optimal solutions for all the 13 problems for which the improvement phase of the PC method stopped by a redundant solution set including the optimal one. Thus we can say that the NBP method is capable of obtaining better solutions than the PC method.

7.2. Evaluation of the computation time of the NBP method

7.2.1. An analytical estimation of the inference time of NBP

As stated in the last section, we can focus on a propagation path for each bubble propagation. The size of this propagation path does not increase with the overall knowledge size. That is, suppose that a couple of sub-BPNs are connected to generate a new and larger one. If we assume that the ratios of the number of ○, □, ● and ■ nodes in both BPNs are equal, then the ratios are also equal in the combined BPN. Since the average size of a propagation path is determined by these ratios—via the probability of extending the propagation path on encountering a ■ or ○ node, the average size of a propagation path is equal in these three graphs (the original pair of BPNs and the resultant BPN after the combination).

This leads to a faster inference by the NBP method than even the PC method, because $O(N^2)$ matrix elements must be computed or checked for equality to 0 in the matrix-based original PC method in each pivoting where N is the number of variables involved (i.e., the number of atoms). In NBP, this $O(N^2)$ time is reduced down to a constant order since each bubble propagation is reduced into a constant-sized propagation path.

The total computation time T_{NBP} can be formulated as

$$T_{NBP} = T_{init} + T_{srch} + T_{impr} \tag{16}$$

where T_{init}, T_{srch} and T_{impr} denote time required for the linear programming, search and improvement phases respectively. T_{srch} is estimated below:

$$\begin{aligned} T_{srch} = a\ &(\text{the number of bubbles which propagate}) \\ &\cdot b\ (\text{average number of source candidates} \\ &\quad \text{in one bubble propagation}) \\ &\cdot c\ (\text{searching time for a bubble's destination} \\ &\quad \propto \text{average size of one propagation path}). \end{aligned} \tag{17}$$

Here, $a \propto N$, since bubble propagation from a ■ node to a ○ node is prevalent and back-propagation (propagation from ● nodes) of bubbles seldom occurs. b is nearly

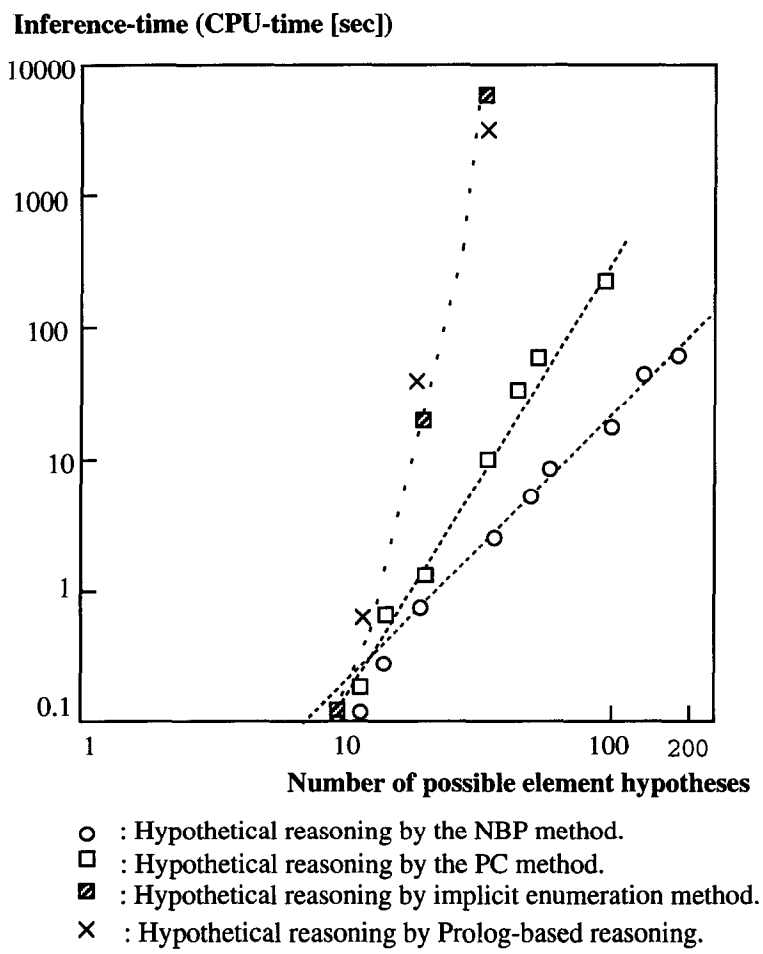


Fig. 7. CPU time of hypothetical reasoning by the NBP and other methods.

proportional to N and c is of constant order if the evaluation above is correct. T_{srch} is thus in the order of $O(N^2)$. In NBP, T_{impr} is in the order of $O(N)$ due to Device 3 of Section 5.2, and the absolute value of T_{init} (time of linear programming) is shorter than T_{srch} and T_{impr}.

As a conclusion, we can estimate T_{NBP} as $O(N^2)$. The main difference between T_{NBP} and the total time $O(N^4)$ of the PC method comes from the difference between the bubble propagation and the matrix calculation of pivoting. This is a merit coming from reflecting the knowledge structure by a network-based mechanism.

7.2.2. Experimental results

Computational speeds of Prolog-based reasoning, an implicit enumeration method [1], the PC method and the present NBP method are depicted in Fig. 7, where each dot

depicts the average inference time of ten problems of a certain problem size. The data of the PC and the NBP methods are close to (or slightly less in the case of NBP) the theoretical estimations of $O(N^4)$ and $O(N^2)$, respectively. Although the Prolog-based and the implicit enumeration methods allow the obtainment of a strictly optimal solution, these two methods cannot finish their computation within a practical or tractable time. The PC and the NBP methods can compute near-optimal solutions in polynomial time; this is obviously important particularly for large problems. Also, jumping to $O(N^2)$ achieved by the NBP method from $O(N^4)$ is a significant improvement for practical adoption of an automatic inference.

8. Conclusions

Recently, the usefulness of local search has been re-recognized for efficient inference as seen in [9,18,23,26]. Unlike these methods, the salient features of the PC method are:

(1) the use of the optimal real domain initial solution obtained quickly by a linear programming method, and
(2) the sophisticated local search mechanism which works in the real domain on the knowledge network rather than just in the binary (0/1) domain.

On the other hand, a network-based approach is important since a knowledge network reflects the structure of the knowledge base. This structure is a fruitful information for fast inference. By merging both merits, i.e., the fast search by the PC method and the network-based inference on the BPN, we achieved $O(N^2)$ time hypothetical reasoning where N is the number of possible hypotheses. This is considerably faster than previous methods.

This improvement in inference time in our new method comes from restricting the network region for each bubble propagating operation (which is functionally equivalent to the pivoting operation). This type of focusing on a limited region in a knowledge base is important in dealing with large-scale knowledge bases.

We are now developing a predicate logic version of NBP which can efficiently deal with predicate Horn clauses with recursion by extending the bubble propagation network (BPN) in a top-down and stepwise expansion manner [19]; this indicates another advantage of our approach of handling networked knowledge.

Appendix A. Propagation rate from a slack node to a structural node

Definition of $p(A, B)$ from $p(B, A)$. For a structural node A and a slack node B pointed from A via an arrow, we define $p(A,B)$ as $p(A,B)p(B,A) = -1$, given the value of $p(B,A)$.

Reason for this definition.. In Fig. 6(2), if the value of S_2 varies by $var(S_2)$, then the variance in V is equal to $p(V,S_2)var(S_2)$. However, since node V stays non-basic from before to after the bubble propagation, it stays in its upper or lower bound, i.e.,

$var(V) = 0$. Thus, the variance at the same time in the value of P must offset this effect from S_2 to V, i.e.,

$$p(V,P)var(P) + p(V,S_2)var(S_2) = 0,$$

i.e.,

$$var(P) = -var(S_2)p(V,S_2)p(V,P).$$

Accordingly, by defining $p(P,V)$ as $p(P,V)p(V,P) = -1$ we obtain $var(P) = p(P,V)p(V,S_2)var(S_2)$. From this equation, the variance in the value of node P is obtained by a calculation along ruling arrows starting from S_2.

Appendix B. Proof of *rank* = *rank'*, at the end of Procedure 3

The number of ○ nodes is equal to the number of ■ nodes (call this number K) because there are the same number of bubbles (● and ■) as the number of structural nodes (○ and ●) in a BPN. Also, by Procedures 1 and 3, each ○ node except $S_1, S_2, \ldots, S_{rank}$ receives one arrow and each ■ node except $T_1, T_2, \ldots, T_{rank'}$ spouts one arrow.

Therefore, $K - rank$ (the number of arrows received by structural nodes) is equal to $K - rank'$ (the number of arrows spouted by slack nodes) because all the arrows spouted from slack nodes are received by structural nodes and those received by structural nodes are spouted by slack ones. It follows that *rank* is equal to *rank'*.

Appendix C. Proofs of the lemmas

Lemma 2. *If a ruling arrow is attached to every edge in a BPN, then all the* ○ *nodes take values of* 0 *or* 1.

Proof. Every node (specifically ○ nodes) in a BPN, all of whose element edges are attached ruling arrows, can be traced back to ● nodes in the opposite direction of the ruling arrows, if there is no loop of arrows (we cannot go out of a loop of arrows by tracing this way). However, a loop L of arrows cannot exist for the following reasons.

(1) If the first node in L to spout arrows to edges in L is a ○ node A, the edges in L which are adjacent to A are all spouted from A.
(2) The first node in L to spout arrows cannot be a ■ node because a ■ node cannot spout an arrow to an edge until it receives arrows from all other edges.
(3) L cannot include a ● or a □ node, because a ● and a □ node is a spouter and a receiver of arrows, respectively.

Therefore, any ○ node's value is ruled by ● nodes via ruling arrows.

Also, if a ■ node P receives (number of P's adjacent edges $-$ 1) ruling arrows, then the value of the single remaining adjacent structural node of P to which a ruling arrow from P outgoes is equal to 0 or 1 because the coefficients of Eq. (7) are all 1 or -1 and the value of each ○ node is restricted to $[0, 1]$.

Thus, by tracing ruling arrows beginning with ● nodes along ruling arrows, we can conclude that all ○ nodes' values are 0 or 1. □

In the proof of Lemma 3, we use Lemma 4 below, so let us precede with the proof of Lemma 4.

Lemma 4. *Every edge in the BPN is attached a ruling arrow if the overall BPN is singly connected.*

Proof. Here we prove that the negation of Lemma 4 is false. That is, we suppose that a BPN is singly connected but includes Γ, a set of edges to which ruling arrows cannot be attached by Procedure 1.

If this assumption is true, there exists a directed subgraph G, which is connected (i.e., any pair of nodes in G are connected by edges in G) even if Γ is pruned off the BPN, and which is connected to only one edge e in Γ. This is because it contradicts with the assumption that the overall knowledge network is singly connected, if all the connected (and directed) subgraphs are connected with each other via more or equal to two edges in Γ, i.e., if a limitless or looped graph is included in the overall BPN.

Thus, all the edges touching a node $P \in e \cap G$ except one (e) are arrowed according to the definition of G. Therefore, whichever ○, □, ●, or ■ node P is, e is arrowed by Procedure 1. This contradicts with the assumption $e \in \Gamma$. □

Lemma 3. *A ruling arrow is attached to every edge in the BPN if the overall knowledge network is singly connected.*

Proof. In the first place, let us classify slack nodes as that nodes S and T_i in Fig. 2 are of type S and T, respectively. For example, in Fig. 3, the thin lined squares are type S and the bold lined are type T.

Now assume that the knowledge network for the given background knowledge is singly connected. If a closed path exists in a BPN for such knowledge, this closed path shares ■ nodes of both types S and T (see Section 4.1 for the definition of these types) with some single AND- or OR-rule R. (This is because a sub-BNP including only one type of slack nodes, S or T, is singly connected, i.e., such a sub-BPN is isomorphic with the given knowledge network which is supposed to be singly connected here.)

Call these ■ nodes of T type and S type (shared between the closed path and R) nodes T and S respectively. Then, $X_1 = Y$ where X_1 and Y are the two adjacent nodes of T, since $T = 0$.

Therefore, the sum of values of all other nodes than X_1 and Y in R (for example, $X_2 + X_3 + \cdots + X_n$ of Fig. 2) is equal to $n - 1$ if $S = 1 - n$ (the lower bound of S in Eq. (5)) and equal to 0 if $S = 0$ (the upper bound of S in Eq. (6)). The only possible values sets of $(X_2, X_3, \ldots, X_n)$ for $S = 1 - n$ and $S = 0$ are $(1, 1, \ldots, 1)$ and $(0, 0, \ldots, 0)$, respectively, because the value of a structural node is in the range of $[0, 1]$. Thus, $X_2, X_3, \ldots, X_n$ are all ● nodes, so that there are at least $(n-1)$ structural nodes $(X_2, X_3, \ldots, X_n)$ and two slack nodes (S and T) which are non-basic, in R. This state of nodes, i.e., $(n + 1)$ non-basic nodes for n structural nodes, is of redundant or

inconsistent constraints, so the assumption that a closed path exists in the overall BPN is negated.

Thus, the overall BPN is singly connected if the overall knowledge network is singly connected. This, combined with Lemma 4, derives the lemma. □

References

[1] E. Balas, An additive algorithm for solving linear programs with zero-one variables, *Opsearch* **13** (1965) 517–546.

[2] E. Balas and C. Martin, Pivot and compliment—a heuristic for 0–1 programming, *Manage. Sci.* **26** (1980) 86–96.

[3] F. Bry, Query evalution in recursive databases, bottom-up and top-down reconciled, *Data Knowledge Eng.* **5** (1990) 289–312.

[4] E. Charniak and S.E. Shimony, Cost-based abduction and MAP explanation, *Artif. Intell.* **66** (1994) 345–374.

[5] G.B. Dantzig, *Linear Programming and Extension* (Princeton University Press, Princeton, NJ, 1963).

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

[7] E.C. Freuder, A sufficient condition of backtrack-free search, *J. ACM* **29** (1982) 24–32.

[8] H. Geffner and J. Pearl, An improved constraint-propagation algorithm for diagnosis, in: *Proceedings IJCAI-87*, Milan (1987) 1105–1111.

[9] J. Gu, Local search for satisfiability (SAT) problem, *IEEE Trans. Syst. Man Cybern.* **23** (1993) 1108–1129.

[10] J.N. Hooker, A quantitative approach to logical inference, *Decis. Support Syst.* **4** (1988) 45–69.

[11] K. Inoue, Y. Ohta and R. Hasegawa, Hypothetical reasoning systems on the MGTP, ICOT Tech. Rept. TR-763 (1992).

[12] M. Ishizuka and F. Ito, Fast hypothesetical reasoning system using inference-path network, in: *Proceedings IEEE Conference on Tools for AI* (1991) 352–360.

[13] 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: *Proceedings 10th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Banff, Alta. (1994) 179–186.

[14] N. Karmarkar, A polynomial-time algorithm for linear programming, *Combinatorica* **4** (1984) 373–395.

[15] H.A. Kautz and B. Selman, Hard problems for simple default logics, in: *Proceedings First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ont. (1989) 189–197.

[16] A. Kondo, T. Makino and M. Ishizuka, An efficient hypothetical reasoning for predicate-logic knowledge-base, in: *Proceedings IEEE International Conference on Tools for AI* (1991) 360–367.

[17] T. Makino and M. Ishizuka, A hypothetical reasoning system with constraint handling mechanism and its applications to circuit-block synthesis, in: *Proceedings Pacific Rim International Conference on Artificial Intelligence '90*, Nagoya (1990) 122–127.

[18] S. Minton, Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, *Artif. Intell.* **58** (1992) 161–205.

[19] Y. Ohsawa and M. Ishizuka, Networked bubble propagation: a polynomial-time predicate-logic hypothetical reasoning by networked bubble propagation method, in: G. McCalla, ed., *Advances in Artificial Intelligence (Proceedings Eleventh Canadian Conference on AI)*, Lecture Notes in Artificial Intelligence **1081** (Springer, Berlin, 1996) 375–387.

[20] Y. Ohta and K. Inoue, A forward-chaining hypothetical reasoner based on upside-down meta-interpretation, in: *Proceedings International Conference on Fifth Generation Computer Systems* (1992) 522–529.

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

[22] D. Poole, R. Aleliunas and R. Goebel, Theorist: a logical reasoning system for defaults and diagnosis, in: N.J. Cercone and G. Macalla, eds., *Knowledge Frontier: Essays in the Knowledge Representation* (Springer, New York, 1987).

[23] B. Selman and H. Kautz, An empirical study of greedy search for satisfiability testing, in: *Proceedings AAAI-93*, Washington, DC (1993) 46–51.

[24] E. Santos Jr, A linear constraint satisfaction approach to cost-based abduction, *Artif. Intell.* **65** (1994) 1–28.

[25] E. Santos Jr and E.S. Santos, Polynomial solvability of cost-based abduction, *Artif. Intell.* **86** (1996) 157–170.

[26] B. Selman and H. Kautz, Domain-independent extensions to GSAT: solving large structured satisfiability problems, in: *Proceedings IJCAI-93*, Chambery (1993) 290–295.

[27] J. Stillman, It's not my default: the complexity of membership problems in restricted propositional default logics, in: *Proceedings AAAI-90*, Boston, MA (1990) 571–578.

[28] L. Vieille, From QSQ towards QoSaQ: global optimization of recursive queries, in: *Proceedings Expert Database Systems* (1988) 421–435.