ELSEVIER

Knowledge-Based Systems 15 (2002) 369–376

Knowledge-Based SYSTEMS

www.elsevier.com/locate/knosys

SL method for computing a near-optimal solution using linear and non-linear programming in cost-based hypothetical reasoning

M. Ishizuka*, Y. Matsuo

Department of Information and Communication Engineering, School of Engineering, University of Tokyo, 7-3-1 Hongo Bunkyo-ku, Tokyo 113-8656, Japan

Received 2 May 2001; accepted 22 November 2001

Abstract

Hypothetical reasoning is an important framework for knowledge-based systems, however, its inference time grows exponentially with respect to problem size. In this paper, we present an understandable efficient method called *slide-down and lift-up* (SL) method which uses a linear programming technique for determining an initial search point and a non-linear programming technique for efficiently finding a near-optimal 0–1 solution. To escape from trapping into local optima, we have developed a new local handler, which systematically fixes a variable to a locally consistent value. Since the behavior of the SL method is illustrated visually, the simple inference mechanism of the method can be easily understood. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Hypothetical reasoning; Linear programming; Non-linear programming

1. Introduction

By handling incomplete hypothesis knowledge which possibly contradicts with other knowledge, hypothetical reasoning tries to find a set of element hypotheses which is sufficient for proving (or explaining) a given goal (or a given observation) [13]. Because of its theoretical basis and its practical usefulness, hypothetical reasoning is an important framework for knowledge-based systems, particularly for systems based on declarative knowledge. However, since hypothetical reasoning is a form of non-monotonic reasoning and thus an NP-complete or NP-hard problem, its inference time grows exponentially with respect to problem size. In practice, slow inference speed often becomes the most crucial problem.

There already exist several investigations trying to overcome this problem. For example, see Refs. [8–12] for authors' work. Besides symbolic inference methods which have been exploited mostly in AI field, search methods working in continuous-value space have recently shown promising results in achieving efficient inference for hypothetical reasoning as well as for SAT problems and constraint satisfaction problem (CSP). This approach is closely related to mathematical programming, particularly with 0–1 integer programming. When we consider a cost-based propositional hypothetical-reasoning problem, it can be transformed into an equivalent 0–1 integer programming problem with a set of inequality constraints. Although its computational complexity still remains NP-hard, it allows us to exploit a new efficient search method in continuous-value space. One key point here is an effective use of the efficient simplex method for linear programming, which is formed by relaxing the 0–1 constraint in 0–1 integer programming. Also, non-linear programming formation provides us another possibility. These approaches may be beneficial particularly for developing efficient approximate solution methods, for example, in cost-based hypothetical reasoning [2].

The pivot-and-complement method [1] is a good approximate solution method for 0–1 integer programming. Ishizuka and Okamoto [9] used this method to realize an efficient computation of a near-optimal solution for cost-based hypothetical reasoning. Ohsawa and Ishizuka [12] have transformed the behavior of the pivot-and-complement method into a visible behavior on a knowledge network, improved its efficiency by using the knowledge structure of a given problem, and consequently developed networked bubble propagation (NBP) method. NBP method can empirically achieve a polynomial-time inference of $N^{1.4}$ where N is the number of possible element hypotheses, to produce a good quality near-optimal solution in cost-based hypothetical reasoning.

The key point of these methods is the determination of an initial search point by using simplex method and efficient

* Corresponding author. Tel.: +81-3-5841-6755; fax: +81-3-5841-8570.

PII: S0950-7051(02)00020-5

local searches around this point in continuous space for eventually finding a near-optimal 0–1 solution. However, in order to avoid trapping into locally optimal points, a sophisticated control of the local search is required; as a result, the inference mechanisms have become complicated and for humans they are difficult to understand.

On the other hand, Gu [5,6] exploited an efficient method to solve SAT problems by transforming them into unconstrained non-linear programming; the SAT problem is related to propositional hypothetical reasoning. There are several methods for unconstrained non-linear programming, i.e. the steepest descent method, Newton's method, quasi-Newton method, and conjugate direction method. The mechanisms of these local search methods are easy to understand as they basically proceed by descending a valley of a function. However, since these methods simply try to find a single solution depending upon an initial search point, they cannot be used for finding a (near) optimal solution for the following reason: if they are trapped into local optima, they have to restart from a new initial point that is to be selected randomly.

In order to find a near-optimal solution using the simple non-linear programming technique, we combine a linear programming, namely simplex method, to determine the initial search point of the non-linear programming. The search, however, often falls into local optima and an effective method escaping from these local optima is required. In this paper, we present an effective method named *variable fixing method* for this problem. Variable fixing corrects a local inconsistency at each locally optimal point and allows to restart the search. Unlike conventional random restart schemes, this method permits to direct the search systematically using the knowledge structure of a given problem.

As sliding-down operations toward a valley of a non-linear function and lifting-up operations are repeated alternately, we call this method *slide-down and lift-up* (SL) method. By illustrating its behavior visually, we show that its mechanism is easily understandable and also achieves good inference efficiency that is close to the efficiency of the NBP method.

In this paper, we will treat hypothetical reasoning problems that are represented in propositional Horn clauses; we will also allow for (in)consistency constraints among hypotheses.

2. Transformation into linear and non-linear programming and their combination

First we show how to transform a hypothetical reasoning problem into linear and non-linear programming problems. These transformations become the basis of the SL method. As for the transformation into linear programming, there are several ways of replacing logical knowledge by an equivalent set of linear inequalities. Among them, we adopt the following transformation used in Ref. [14].

Associating the `true/false` states of logical variables such as $p1$, $p2$, q, etc. with 1/0 of the corresponding numerical variables represented by the same symbols, we transform a Horn clause

$$q \leftarrow p1 \wedge p2 \tag{1}$$

into a set of inequalities

$$q \leq p1, \qquad q \leq p2, \qquad p1 + p2 - 1 \leq q \tag{2}$$

and also

$$q \leftarrow p1 \vee p2. \text{ (combination of } q \leftarrow p1 \text{ and } q \leftarrow p2) \tag{3}$$

into

$$p1 \leq q, \qquad p2 \leq q, \qquad q \leq p1 + p2. \tag{4}$$

This transformation is advantageous in that it allows to produce a 0–1 solution only by using simplex method for a certain type problem [15], though the number of generated inequalities becomes large.

For the constraint representing inconsistency, the head of its Horn clause is set to `false` which is translated to 0 in the corresponding inequality. As the goal of hypothetical reasoning has to be satisfied, it is set to `true` which becomes 1 in the inequality.

Let the weights of possible element hypotheses $h1, h2, h3, \ldots$ be $w_1, w_2, w_3, \ldots$, respectively. Moreover, let the element hypothesis hi ($i = 1, 2, \ldots$) become $hi = 1$ if it is included in the solution hypothesis, and $hi = 0$ otherwise. Then we can define the cost of the solution as

$$\text{cost} = w_1 h1 + w_2 h2 + w_3 h3 + \cdots$$

which expresses the sum of the weights of the element hypotheses included in the solution. Let us set cost as the objective function; then if we compute the optimal solution to minimize this function under the generated inequalities, it indicates the optimal solution in the cost-based hypothetical reasoning problem.

In this way, hypothetical reasoning becomes a 0–1 linear integer programming. In the pivot-and-complement method (Balas 80), an efficient approximate solution method for 0–1 integer programming, the optimal real-number solution is obtained as follows: first the simplex method is used by relaxing the 0–1 constraint on the variables, then a near-optimal 0–1 solution is searched in a sophisticated manner around the optimal real-number solution. This local search mechanism for the 0–1 solution is rather complicated because it incorporates several heuristics that have been obtained empirically. For our new method here, while we utilize this optimal real-number solution obtained by the simplex method as the initial search point, we try to develop a new simple and understandable method of the local search using a non-linear programming technique.

Gu [5,6] presented a method for SAT problems by transforming them into unconstrained non-linear programming

problems. According to this transformation, the transformation of propositional Horn clauses to clauses that can be processed by our hypothetical reasoning can be realized as follows.

- If the same variable appears in head part of a clause, we introduce new variables and produce a rule clause with disjunctive body as: In case of $q \leftarrow p1 \wedge p2$, and $q \leftarrow p3 \wedge p4$, replace them by $q1 \leftarrow p1 \wedge p2$, $q2 \leftarrow p3 \wedge p4$, $q \leftarrow q1 \vee q2$. (This is to make the following completion operation effective even for OR-related rules.)
- Apply completion for each rule. (Completion is an operation that changes $q \leftarrow p$ to $q \leftarrow p$ and $q \rightarrow p$, interpreting $\{q$ if $p\}$ as $\{q$ if and only if $p\}$. In the definition of material implication (used in formal logic), $q \leftarrow p$, for instance, is interpreted as `true` if q is `true` regardless of ps truth value; this makes backward inference impossible which is required in the hypothetical reasoning. Thus completion is needed here.)

After the above operations, a given problem is transformed to the problem of finding the minimal value 0 of a non-linear function constructed as follows.

- Associate the `true/false` states of each logical variable with $1/-1$, respectively, of the corresponding numerical variable.
- Replace the literals x and $\neg x$ by $(x-1)^2$ and $(x+1)^2$, respectively.
- Replace conjunction ($\wedge$) and disjunction ($\vee$) in the logical formula by arithmetic operation $+$ and $\times$, respectively, assuming that all the clauses are connected by conjunction.

For example, consider the following hypothetical reasoning problem.

$1 \leftarrow g$. (The goal g shall be satisfied.)
$g \leftarrow a \wedge b.$, $a \leftarrow h1 \wedge c.$, $b \leftarrow h2 \wedge c.$, $c \leftarrow h3 \wedge h4.$,
$c \leftarrow h5 \wedge h6.$, inc $\leftarrow h1 \wedge h4$.

'inc' stands for inconsistent which is equivalent to the empty clause.

After applying completion, the following set of formulae is generated.

$1 \leftarrow g.$, $g \leftarrow a \wedge b.$, $a \leftarrow g.$, $b \leftarrow g.$, $a \leftarrow h1 \wedge c.$, $h1 \leftarrow a.$, $c \leftarrow a.$, $b \leftarrow h2 \wedge c.$, $h2 \leftarrow b.$, $c \leftarrow b.$, $c1 \vee c2 \leftarrow c.$, $c \leftarrow c1.$, $c \leftarrow c2.$, $c1 \leftarrow h3 \wedge h4.$, $h3 \leftarrow c1.$, $h4 \leftarrow c1.$, $c2 \leftarrow h5 \wedge h6.$, $h5 \leftarrow c2.$, $h6 \leftarrow c2.$, inc $\leftarrow h1 \wedge h4$.

Then the following non-linear function is to be constructed, for which a solution achieving the minimal value 0 is searched.

$$
\begin{aligned}
f = {} & (g-1)^2 + (g-1)^2(a+1)^2(b+1)^2 + (g+1)^2(a-1)^2 \\
& + (g+1)^2(b-1)^2 + (a-1)^2(h1+1)^2(c+1)^2 \\
& + (a+1)^2(h1-1)^2 + (a+1)^2(c-1)^2 \\
& + (b-1)^2(h2+1)^2(c+1)^2 + (b+1)^2(h2-1)^2 \\
& + (b+1)^2(c-1)^2 + (c+1)^2(c1-1)^2(c2-1)^2 \\
& + (c-1)^2(c1+1)^2 + (c-1)^2(c2+1)^2 + (c1-1)^2 \\
& \times (h3+1)^2(h4+1)^2 + (c1+1)^2(h3-1)^2 \\
& + (c1+1)^2(h4-1)^2 + (c2-1)^2(h5+1)^2(h6+1)^2 \\
& + (c2+1)^2(h5-1)^2 + (c2+1)^2(h6-1)^2 \\
& + (h1+1)^2(h4+1)^2. \qquad (5)
\end{aligned}
$$

The weight of each element hypothesis is not considered here; thus, even if a 0–1 solution achieving $f = 0$ is found, it will not necessarily be an optimal or near-optimal one. However, if local search is conducted starting from the real-value optimal point determined by the simplex method with respect to the linear inequality constraints and the objective function, it will reach a near-optimal 0–1 solution.

One feature of the non-linear function thus generated is that it is at most quadric with respect to one variable. This is due to the fact that one variable never appears twice in one propositional Horn clause.

3. Improvement of the construction of the non-linear function

Although the above computational mechanism using both the linear and non-linear programming techniques provides us with an understandable scheme for cost-based hypothetical reasoning, we have to solve one crucial problem, namely trapping into local optimal points where f is strictly above 0. To cope with this problem, we first reconsider the form of the above non-linear function.

The following example shows a simple case of trapping into a local optimal point

$$a(\texttt{false}) \leftarrow b(\texttt{true}) \wedge c(\texttt{false}) \wedge d(\texttt{true}) \qquad (6)$$

$$c(\texttt{false}) \leftarrow e(\texttt{false}) \vee p(\texttt{true}) \qquad (7)$$

where `true/false` inside ( ) indicates the truth value of the variable in a state. In this example, Eq. (7) is in a false state; this prevents the non-linear function covering all the logical formulae or constraints to become 0. After completing Eqs. (6) and (7), the non-linear function obtained as a result of the proposed replacements becomes Eqs. (8) and

(9), respectively:

$$(a-1)^2(b+1)^2(c+1)^2(d+1)^2+(a+1)^2(b-1)^2 + (a+1)^2(c-1)^2+(a+1)^2(d-1)^2 \quad (8)$$

$$(c+1)^2(e-1)^2(p-1)^2+(c-1)^2(e+1)^2 + (c-1)^2(p+1)^2. \quad (9)$$

Suppose that c appears in these two clauses of the knowledge base. Then, to consider the effect of cs change, we compute the partial derivatives of Eqs. (8) and (9) with respect to c as

$$(-1-1)^2(1+1)^2(2(c+1))(1+1)^2+(-1+1)^2(2(c-1)) = 128(c+1) \quad (10)$$

$$2(c+1)(-1-1)^2(1-1)^2+2(c-1)(-1+1)^2+2(c-1) \times (1+1)^2 = 8(c-1) \quad (11)$$

As these are summed to compute $\partial f/\partial c$ as in Eq. (5), we obtain $\partial f/\partial c = 136c + 120$. Since f is quadratic with respect to one variable, f becomes minimal at $c = -120/136$. This suggests that c tends to stick to -1(`false` state).

This example indicates that the non-linear function proposed by Gu [5,6] is problematic when a parent node[1] has many child node[1] and the truth status of a clause is determined by its one child node. That is, since the coefficient of the variable corresponding to this node increases with the involution of 2 in the non-linear function, as seen in Eq. (8), this effect becomes too large so as to prevent the same variable appearing in the head part of another Horn clause moving away from its incorrect status. The problem here is that if p is `true`, then $(p+1)^2$ becomes 2^2; that is, the coefficient of one variable in a certain state varies with the involution of 2, depending on the number of body atoms in the Horn clause.

Hence, rather than associating `true/false` states with $1/-1$, respectively, we associate them with $0.5/-0.5$, respectively, and transform x and $\neg x$ to $(x-0.5)^2$ and $(x+0.5)^2$, respectively, to construct a new non-linear function. Thereby, when p is `true`, $(p+0.5)^2$ becomes 1; the unbalance among the coefficients of the product terms in a certain state can be avoided.

The following list summarizes the way of constructing the new non-linear function.

- Associate the `true/false` states of each logical variable with $0.5/-0.5$, respectively, of the corresponding numerical variable.
- Replace the literals x and $\neg x$ by $(x-0.5)^2$ and $(x+0.5)^2$, respectively.
- Replace conjunction ($\wedge$) and disjunction ($\vee$) in the logical formula by the arithmetic operations $+$ and $\times$, respectively.

[1] In this paper, 'variable' and 'node' are used interchangeably since a propositional variable constitutes one node in the proof tree.

4. Variable fixing scheme for escaping from local optima

There remains the possibility that the search is trapped into local optima and cannot go to the minimal point of $f = 0$ even if we use the new non-linear function. For example, consider the following simple case. If all the values are `false` (namely, -0.5) at the initial stage, this becomes a locally optimal point. As $h1$, $h2$, $h3$ and $h4$, which are all in a `false` state, pull a and b toward their `false` states, the goal g cannot go to a `true` state.

$$g \leftarrow a \wedge b., \qquad a \leftarrow h1 \wedge h2., \qquad b \leftarrow h3 \wedge h4.$$

In large complex problems, the non-linear functions tend to have many locally optimal points of this and other sorts.

To solve this problem, consider again the following case of a locally optimal state.

$$a(\texttt{false}) \leftarrow b(\texttt{true}) \wedge c(\texttt{true}) \wedge d(\texttt{false}) \quad (12)$$

$$d(\texttt{false}) \leftarrow e(\texttt{true}) \wedge p(\texttt{true}) \wedge q(\texttt{true}). \quad (13)$$

When the non-linear function does not reach 0, the minimum value, it indicates that at least one Horn clause is not satisfied. In this example Eq. (13) is not satisfied, since two restrictions, one from an upper node (d shall be `false`) and one from a lower node (d shall be `true`), contradict each other. This situation cannot simply be resolved because a might be requested to be `false` from its upper node, and also e, p and q might be requested to be `true` from their lower nodes.

We thus determine the state of d to be `true` or the state of either e, p or q to be `false`, so as to escape from the locally optimal point. There are two cases in unsolved states where the goal is unsatisfied: first, some of the element hypotheses which are needed to prove the goal are missing, and secondly, selected hypotheses violate the consistency constraint. Since the former case arises more often empirically in the search process starting from the initial point, which is obtained as the real-number optimal point by the simplex method, we adopt the strategy of resolving the former case first. In this case, if the restrictions from the upper node and from the lower node contradict, the central node is set to `true`.

In addition, after restarting from a new state, the search often falls into the same local optimal point since the non-linear function is quadratic with respect to one variable. Thus, if one variable is changed to 0.5 (`true` state) to resolve the local inconsistency, we do not change it anymore in the search process; that is, we consider it a constant rather than a variable afterwards. As this implies

the substitution of 0.5 in a strong sense, we call this operation *variable fixing*. The effect of variable fixing propagates to other variables through the minimization process of the non-linear function.

The target nodes (variables) for the variable fixing operations are as follows. Here, we evaluate the state of each node by rounding its value to -0.5(`false`) or 0.5(`true`).

1. The goal node in a `false` state.
2. A parent node in a `false` state such that its AND-related child nodes are all in a `true` state.
3. A parent node in a `false` state such that one of its OR-related child node is in a `true` state.
4. An AND-related child node in a `false` state such that its parent node is in a `true` state.
5. One of the OR-related child nodes in a `false` state such that its parent node is in a `true` state.

When fallen into a locally optimal point, we find one of the target nodes (variables) as listed above sequentially in order of numbers (1)–(5), apply the variable fixing operation to that variable and restart the search. The reason for applying variable fixing to one target variable at a time rather than to multiple target variables is to avoid moving apart far from the real-number optimal point which is obtained by the simplex method.

In (2) and (3) above where the restriction from the lower nodes is considered, variable fixing does not contribute to satisfy the goal and hence causes inconsistency in rare case, since the states of the lower nodes are determined by considering the consistency constraints.

On the other hand, in (4) and (5) above, although variable fixing may contribute to satisfy the goal, it possibly causes inconsistency by changing a `false` child node to `true`. Thus, we place priority on (2) and (3) over (4) and (5).

In (5), the selection of one target node is not uniquely determined; however, we select the one with the largest analog value as it may affect the states of its lower nodes the least. We call this process an 'OR-node selection phase'.

Experiments show that the order of (2), (3) and (4) above causes no big difference on the performance. The algorithm always stops in finite steps, since one variable fixing operation always decreases one free variable. Also, as the variable fixing operation increases the number of `true` nodes, it guarantees to produce a solution if the given problem has no consistency constraint. Redundant element hypotheses in the solution can be removed in an improvement phase in order to obtain a near-optimal solution.

Yet, there are cases where a solution cannot be obtained due to variable fixing of wrong nodes (variables) which contradict with the consistency constraint. These cases arise the following situations.

(i) The node which shall be `false` is fixed to `true`.
(ii) In the OR-node selection phase one wrong node, which is different from one to be `true`, is fixed to `true` and causes inconsistency.

For case (i), try to remove the redundant element hypotheses from the resulting solution hypothesis set so that the inconsistency can be resolved (redundancy detection phase). For case (ii), keep other possible OR-related child nodes not selected in a stack, and if the search is failed, restart the OR-node selection phase by picking another child node from the stack. Although this operation causes inefficient backtracking, it almost never occurs in our experiments and therefore does not become a serious problem.

In our system, the OR-nodes are usually selected in the minimization process of the non-linear function. Here, if a wrong OR-node is selected, there is no way to remedy it. Then the search proceeds via the local optima by the variable fixing operations; redundant element hypothesis in the solution hypothesis can be removed in the improvement phase described in Section 5.

The variable fixing operation is effective only when the new non-linear function introduced in this paper is used. If Gu's non-linear function is used, it is not useful since the probabilities of the wrong selection in the OR-node selection phase and of fixing wrong nodes in other steps become large due to the coefficient unbalance of the terms in the non-linear function.

5. Improvement phase of solution cost

The solution obtained after the search undergoes an improvement phase taking into account its cost. A simple algorithm is employed here; that is, each element hypothesis included in the solution hypothesis set is checked in order according to its weight to see whether or not it can be removed without undercutting the provability of the goal. If it can, then the element hypothesis is removed from the solution. This operation, which is also employed in the pivot-and-complement method [1] for 0–1 integer programming, seems to be the most effective in terms of its computational cost and its performance.

6. Algorithm of SL method

Summarizing above descriptions, we show below the algorithm of the SL method for cost-based hypothetical reasoning. The steepest descent method is currently employed for the minimization of the non-linear function.

Initial phase: Constitute a set of linear inequalities from a given problem. Relaxing the 0–1 constraint, obtain the optimal real-number solution by the simplex method and set this as the initial search point for the subsequent 0–1 solution search. (The range of [0,1] of the variables at this initial search point is shifted to $[-0.5,0.5]$ when used in the subsequent minimization of the non-linear function.)

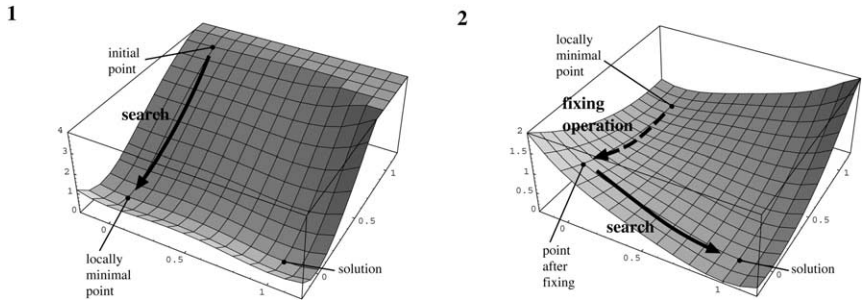

Fig. 1. A behavior of search (example 1).

Constitution of non-linear function: Constitute the non-linear function as described earlier.

Search phase: Execute the search for finding the minimal value 0 of the non-linear function. If rounding the variables into 0.5/−0.5 makes the non-linear function be 0, then stop and go to (6). If a local optimum is detected, then go to (4).

Variable fixing: In order to escape from the local optimum, select one target node according to the order of (1)–(5) described in Section 4, apply variable fixing to this node,

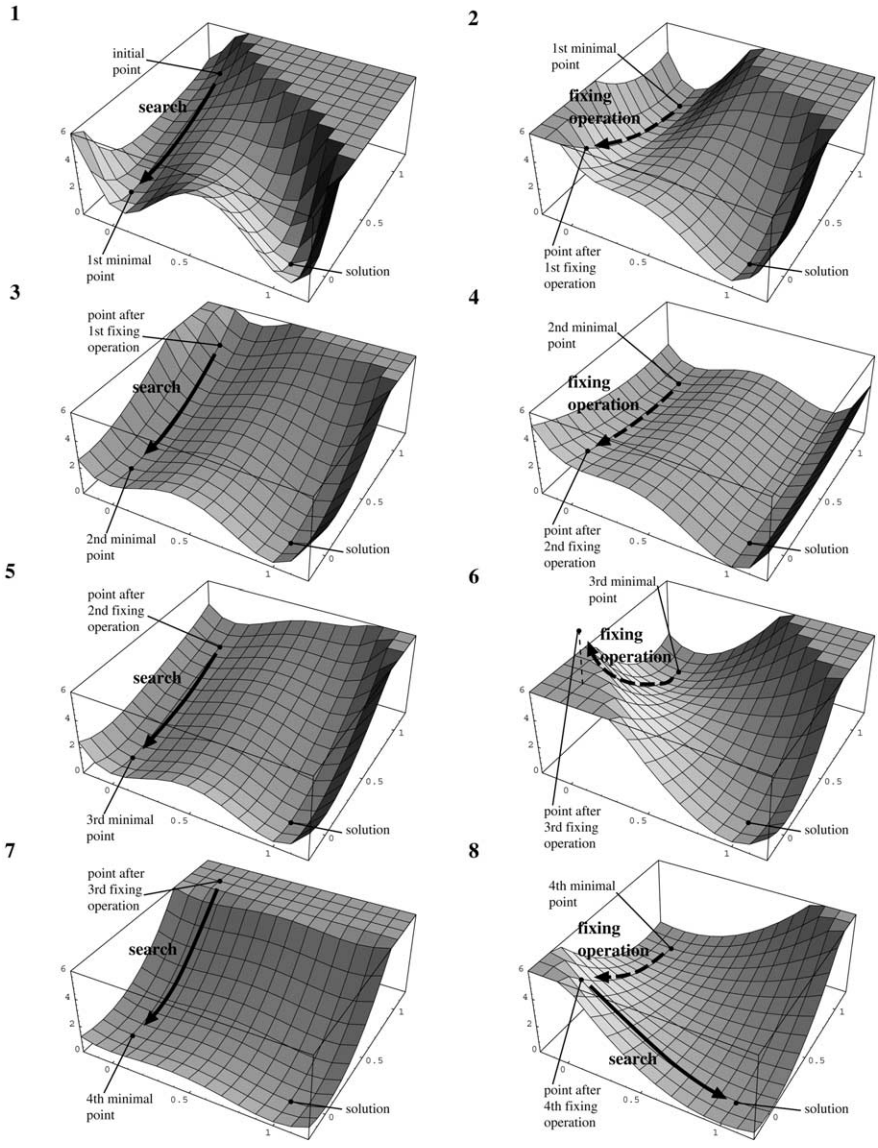

Fig. 2. A behavior of search (example 2).

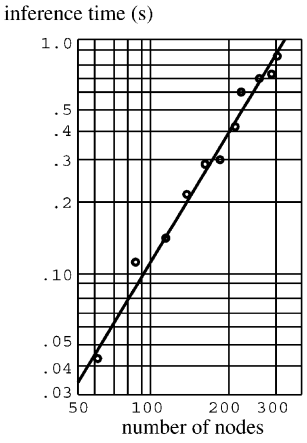

Fig. 3. Inference time of SL method.

and go back to (3). If there is no target node found, then go to (5).

Redundancy detection phase: If there is no target node in (4), it means that all the Horn clauses being related to the proof of the goal are satisfied except the consistency constraints. Thus, keeping the goal node `true`, try to reduce the `true` hypothesis so as to remove the inconsistency among hypotheses. If this succeeds, go to (6). Otherwise, restart from the OR-node selection phase in (4) by picking up one of the remaining OR-nodes in the stack. If there is no such OR-node in the stack, the search fails.

Improvement phase: Try to temporally change each `true` element hypothesis to `false` without spoiling the proof of the goal. If this succeeds, then change this element hypothesis and its associated intermediate nodes to `false`; that is, remove this element hypothesis from the solution hypothesis as a redundant one.

7. Visual illustration of the behavior of the SL method

Unlike other efficient methods for (cost-based) hypothetical reasoning, one salient feature of the SL method is its simple and understandable mechanism, which is basically based on the minimization of the non-linear function starting from the initial search point which is determined by the simplex method. Here, we illustrate its behavior visually.

For visual illustration, we have to find an effective way of mapping the behavior in multi-dimensional space into two or three-dimensional space; simple display methods such as one using two axes of two variables are not good enough. For effective display, we choose here two axes such that one axis corresponds to the vector from the current search point to the previous point determined by variable fixing or the minimization process, and another corresponds to the vector from the current search point to the optimal solution point. We set the current search point, the previous point and the optimal solution point at (0,0), (0,1) and (1,0), respectively, and that the height represents the shape of the non-linear function.

Fig. 1 shows an example in which a solution with function value 0 can be found after only one variable fixing operation. Starting from an initial search point, the search first falls into a locally minimal point, from which it cannot proceed to the solution point as there is a small hill between them. One can observe from the figure on the right that variable fixing changes the form of the function and allows the search to go to the solution point.

Fig. 2 shows a more complicated case, in which four variable fixing operations are required before reaching the solution. It can be seen well that the minimization (Slide-down) and variable fixing (Lift-up) operations are repeated. (These illustrations are produced using Mathematica.)

We want to emphasize that this visualization facilitates comprehension of the simple inference mechanism of the SL method; the simplicity is important to grasp the inference behavior and to trust it. For a computer, visualization is not necessary to execute the search.

8. Experimental results and evaluation

Fig. 3 shows the experimental results of the inference time of the SL method. The system is implemented in C and runs on Sun Ultra and SGI workstations. The number of body atom in each Horn clause is 2–7, and the number of the occurrences of each atom in one experimental knowledge set is at most 10. The horizontal axis is the number of nodes, as the SL method carries out the search on the nodes. Since in practice, the simplex method produces its optimal solution in polynomial-time, the average data excluding cases in which the 0–1 solutions were obtained only using simplex method are plotted in Fig. 3. Also, the times spent for the simplex method are excluded from the plotted data.

As shown in Fig. 3, the SL method can compute a near-optimal solution in a polynomial-time of approximately $n^{1.8}$ where n is the number of nodes. Failure of finding a solution was one case out of 111 problems. The near-optimality of the obtained solution is as good as that of the NBP method [12], in which at least the third nearest solution to the optimal one is obtained for all the cases where the system finds the solution.

9. Related work

ATMS [3] is an efficient method for dealing with possibly hypothetical knowledge. It is used as a caching mechanism for the working memory of a production system and to maintain inferred data that are supported by consistent hypotheses (‘label’ in ATMS terminology). Although its mechanism is also useful partially for logical problem solving, it is not enough for problem solving in general. The logic-based framework of hypothetical reasoning was presented in Ref. [13]. Since then, several schemes have

been developed to improve its slow inference speed. A method using an inference-path network [8] where network compilation phase is based on the linear-time algorithm of Dowling and Gallier [4] for propositional Horn logic, has achieved nearly ultimate speed for propositional hypothetical reasoning. As for predicate-logic hypothetical reasoning, effective fast inference methods have been presented in Refs. [10,11], which use a deductive database technique.

On the other hand, the effectiveness of using mathematical programming techniques for logical problem solving has been advocated by Ref. [7]. Recently, Santos [14,15] showed that many problems of cost-based hypothetical reasoning (or abduction) [2] can be solved using only the simplex method; however, the hypothetical reasoning problem there does not permit any inconsistency among possible hypotheses. Cost-based hypothetical reasoning is closely related to 0–1 integer programming. Both are NP-hard problem in general; however, there exists a good approximate solution method called pivot-and-complement method [1] for 0–1 integer programming. This method was applied in Ref. [9] for computing a near-optimal solution in polynomial-time with respect to problem size to cost-based hypothetical reasoning. In NBP method [12], the behavior of the pivot-and-complement method has been transformed into a visible behavior on a knowledge network and the efficiency has been improved by using the knowledge structure of a given problem. Although the NBP method can achieve high efficiency, i.e. a low-order polynomial-time inference, its mechanism has become complicated.

As another approach of applying mathematical programming techniques, Gu [5,6] exploited the use of unconstrained non-linear programming techniques for SAT problems, which are closely related to logical inference and CSP. These techniques provide simple efficient mechanisms for finding a single solution. Yet it is not appropriate to find the (near) optimal solution, for example, in cost-based hypothetical reasoning.

10. Conclusion

We have presented the SL method for cost-based hypothetical reasoning, which can find a near-optimal solution in polynomial-time in cost-based hypothetical reasoning. It uses both linear and non-linear programming techniques. Moreover, it incorporates a local handler to escape from trapping into locally optimal points. The notable feature of this method is its simple and understandable search behavior as visually illustrated here, though at present its speed performance is slightly lower than that of the NBP method [12]. The mechanism of the SL method may also be useful to develop systematic polynomial-time methods for finding near-optimal solutions in other problems such as the constraint optimization problem.

Acknowledgements

The authors are grateful to Dr Helmut Predinger (University of Tokyo) for his helpful comments on this paper.

References

[1] E. Balas, C. Martin, Pivot and complement—A heuristic for 0–1 programming, Mgmt Sci. 20 (1980) 86–96.
[2] E. Charniak, S. Shimony, Probabilistic semantics for cost based abduction, Proc. AAAI-90 (1992) 106–111.
[3] J. deKleer, An assumption-based TMS, Artif. Intell. 28 (1986) 127–162.
[4] W.F. Dowling, J.H. Gallier, Linear-time algorithm for testing satisfiability of propositional Horn formulae, J. Logic Progm 3 (1984) 267–284.
[5] J. Gu, Local search for satisfiability (sat) problem, IEEE Trans. Syst. Man Cybernetics 23 (4) (1993) 108–1129.
[6] J. Gu, Global optimization for satisfiability problem, IEEE Trans. Knowledge Data Engng 6 (3) (1994) 361–381.
[7] J.N. Hooker, A quantitative approach to logical inference, Decision Support Syst. 4 (1988) 45–69.
[8] M. Ishizuka, F. Ito, Fast hypothetical reasoning system using inference-path network, Proc. IEEE Int. Tools AI (TAI'91) (1991) 352–359.
[9] M. Ishizuka, T. Okamoto, A polynomial-time hypothetical reasoning employing an approximate solution method of 0–1 integer programming for computing near-optimal solution, Proc. Can. Conf. AI (1994) 179–186.
[10] A. Kondo, T. Makino, M. Ishizuka, Efficient hypothetical reasoning system for predicate-logic knowledge-base, Knowledge-Based Syst. 6 (2) (1993) 87–94.
[11] A. Kondo, M. Ishizuka, Efficient inference method for computing an optimal solution in predicate-logic hypothetical reasoning, Knowledge-Based Syst. 9 (1996) 163–171.
[12] Y. Ohsawa, M. Ishizuka, Networked bubble propagation: A polynomial-time hypothetical reasoning for computing near optimal solutions, Artif. Intell. 91 (1) (1997) 131–154.
[13] D. Poole, A logical framework for default reasoning, Artif. Intell. 36 (1988) 27–47.
[14] E. Santos Jr., A linear constraint satisfaction approach to cost-based abduction, Artif. Intell. 65 (1994) 1–27.
[15] E. Santos Jr., E.S. Santos, Polynomial solvability of cost-based abduction, Artif. Intell. 86 (1996) 157–170.