Fast Hypothetical Reasoning by Parallel Processing

Yutaka Matsuo and Mitsuru Ishizuka

Dept. of Information and Communication Eng.
School of Engineering, University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
E-mail: {matsuo,ishizuka}@miv.t.u-tokyo.ac.jp

Abstract. Hypothetical reasoning is an important framework for knowledge-based systems because it is theoretically founded and useful for many practical problems. Since the inference time of hypothetical reasoning grows exponentially with respect to problem size, its inefficiency becomes the most crucial problem when applied to practical problems.
In this paper, we develop a new framework for hypothetical reasoning that uses parallel software processors. Our earlier SL method, which can find a near-optimal solution for cost-based hypothetical reasoning in polynomial time (with respect to problem size), uses both linear programming and nonlinear programming techniques. In the new method, these techniques are realized as the interaction of parallel processors. Taking this approach, we may generalize related methods such as the breakout method or Gu's nonlinear optimization method for SAT problems, and introduce two superior algorithms. One algorithm is similar to the breakout method, and the other achieves good-quality solutions by adding new processors during search iterations.

Keywords. hypothetical reasoning, parallel computation, near-optimal solution

1 Introduction

Hypothetical reasoning tries to find a set of element hypotheses that is sufficient for proving (or explaining) a given goal (or a given observation) [Poole 88]. The assumption of hypothetical knowledge allows to handle domains where only incomplete knowledge is provided. Hypothetical reasoning can also deal with cases where hypothesis knowledge contradicts with domain knowledge. Because of its theoretical basis and practical usefulness, hypothetical reasoning is an important framework for knowledge-based systems. 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. Slow inference speed often becomes the most crucial problem in practice.

For the related satisfiability (SAT) problems, many kinds of methods based on GSAT have seen great success. One of the strategies to enhance the power and applicability of GSAT is to add a weight to each clause and increase the weight if the clause is not satisfied [Selman and Kautz 93]. The same idea is applied in the breakout method [Morris 93], the heuristic repair method [Minton *et al.* 92], the DLM (discrete Lagrangian based search method) [Wah and Shang 97], the GLS (Guided Local Search) method [Voudouris 95], and others. Those methods show good performance in several domains. The main idea common to those breakout-type methods is the use of constraint weighting schemes that solve the problem of local minima by adding weights to the cost of violated constraints. These weights permanently increase the cost of violating a constraint, thus changing the shape of the cost surface so that (local) minima can be avoided or exceeded [Thornton and Sattar 99].

Ishizuka and co-workers introduced an efficient and comprehensive method called SL (*Slide-down and Lift-up*) method for cost-based hypothetical reasoning. The SL method uses a linear programming technique, namely the simplex method, for determining an initial search point and a non-linear programming technique for efficiently finding a near-optimal 0-1 solution [Ishizuka and Matsuo 98]. The nonlinear programming method is based on Gu's method which applies unconstrained nonlinear optimization for SAT problems [Gu 94]. Starting the search from the real-number optimal solution, the SL method has been experimentally shown to find a near-optimal 0-1 solution in polynomial time with respect to problem size. One salient feature of the SL method is its search in continuous-value space.

In this paper, we introduce a new method that treats each variable and each constraint of a hypothetical reasoning problem(HRP) as a processor. Thus the search for a solution is realized as the interaction of multiple processors. Actually, both linear and nonlinear programming techniques can be built up as the interaction of processors. Starting from a description of the parallel computation method, we show how to design a processor, i.e., how to update its value and send messages to the neighboring processors. By simplifying the message, the relations to other algorithms are shown, including the breakout-type method and Gu's nonlinear optimization method. Finally, two methods are introduced that have very good performance when applied to HRPs. One is similar to breakout method, which increases the weights of violated constraints. The other is a new algorithm, where two different processors work together. When comparing this algorithm to other methods, it can be shown to find qualitatively better solutions.

The rest of this paper is organized as follows. In Section 2, we describe how to transform HRPs into mathematical formulations. Section 3 starts with a brief introduction of parallel computation, and then applies this paradigm to HRPs. In Section 4, the relations to the breakout-type method and Gu's nonlinear optimization method are discussed. Section 5 is dedicated to a new algorithm, called *leastEQ method*, that uses two different types of processors. In Section 6, we evaluate the algorithms experimentally. Section 7 concludes the paper.

2 Transformation into Linear and Nonlinear Programming

The discussion in this paper is restricted to hypothetical reasoning problems that can be represented as propositional Horn clauses. Note that we explicitly include inconsistency constraints—denoting inconsistencies among hypotheses—as special cases of Horn clauses.

In general, a HRP is characterized by a goal g to be explained, given a logical theory Σ modeling some domain. A *solution* to a HRP is a set of hypotheses which, if assumed, would explain g. The set of hypotheses H is typically restricted to some set $\mathcal{H}$ of 'assumable' predicates. More formally, given a HRP, a set $H \subseteq \mathcal{H}$ is a solution for a HRP if and only if (i) $\Sigma \cup H \vdash g$, and (ii) $\Sigma \cup H \not\vdash \bot$, where $\bot$ denotes the impossible state (*falsum*).

First we show how to transform a HRP into linear and nonlinear programming problems, by transforming Horn clauses into equalities and inequalities. In the following, we associate the `true`/`false` states of logical variables i with numerical values 1/0 of the corresponding numerical variables represented by x_i.

Transformation into Linear Programming. The objective function being minimized is

$$f = \sum_{i \in N} w_i x_i, \tag{1}$$

where w_i represents the weight of element hypothesis i, and N represents the set of logical variables. A Horn clause of the form "$a \leftarrow b \vee c$" is transformed into an inequality "$x_a \geq x_b + x_c$", and "$a \leftarrow b \wedge c$" into the inequalities "$x_a \geq x_b$" and "$x_a \geq x_c$"[1]. The linear

[1] After applying completion for each rule [Ishizuka and Matsuo 98], we can eliminate the top-down constraints, as shown by [Santos 94]. The following formulations of nonlinear optimization and equalities are also based on this idea.

inequality corresponding to Horn clause j is

$$g_j^{LI}(x) \leq 0. \tag{2}$$

This inequality expresses a necessary condition for the original Horn clause to be satisfied.

Then we relax the 0-1 constraint on the variables and allow the variables to be in [0,1]. Using the simplex method, the optimal real-number solution is obtained. This optimal real-number solution contains valuable information: (i) the cost of the solution shows the lower bound of 0-1 optimal solutions; (ii) the solution provides a guide to find a near-optimal 0-1 solution; (iii) if the linear programming problem is infeasible, the original HRP is also infeasible.

Transformation into Nonlinear Programming. Gu presents a method for SAT problems by transforming them into unconstrained nonlinear programming problems [Gu 94]. Inspired by Gu's method, a given problem is transformed to the problem of finding the minimal value 0 of a nonlinear function that is constructed as follows.

- Replace the literals x_i and $\neg x_i$ by x_i^2 and $(1-x_i)^2$, respectively.
- Replace conjunction ($\wedge$) and disjunction ($\vee$) in the logical formula by the arithmetic operations $+$ and $\times$, respectively. We assume that all clauses in the knowledge base are (implicitly) connected by conjunction.

For instance, "$1 \leftarrow g$, $g \leftarrow a \vee b$" is transformed into

$$f^{NLP} = (1-x_g)^2 + x_g^2(1-x_a)^2(1-x_b)^2.$$

The term corresponding to Horn clause j in f^{NLP} is denoted by f_j^{NLP}.

Transformation into Equalities. In order to prepare the discussion in the next section, Horn clauses are transformed into equations as follows. For each Horn clause, replace the literals x_i and $\neg x_i$ by x_i and $1 - x_i$, respectively. Replace disjunction by the arithmetic operation $\times$ and construct the left-hand side of the equation. Put 0 to the right-hand side of the equation. For example, "$1 \leftarrow g, g \leftarrow a \vee b$" is transformed into the equalities "$1 - x_g = 0$" and "$x_g(1-x_a)(1-x_b) = 0$".

The equality corresponding to Horn clause j is

$$h_j^{EQ}(x) = 0. \tag{3}$$

If we round the variables to 0/1, this constraint is a necessary and sufficient condition to satisfy the original Horn clause.

3 Parallel Computation

A hypothetical reasoning problem is expressed by a network as shown in Fig. 1. The search for the transformed linear programming problem or nonlinear programming problem described in the previous section is realized by the message-passing in this network. Here, the contents of the message is simply a value.

For example, Fig. 2 shows how to realize nonlinear optimization by the message-passing. Here, each variable and each constraint sends messages. Using $\partial f_j^{NLP}/\partial x_i$, each variable updates its value so that f^{NLP} is minimized.

Thus, we consider each variable and each constraint as a processor. Each processor performs a very simple computation, it just sends a value as a message. Search proceeds as the interaction iterates. Note that we employ the parallel processor model in order to grasp the procedure of the algorithm intuitively. The computation is executed on a single serial machine. We ignore the cost of communication among processors, and assume that each processor can request a message from its neighbors whenever needed.

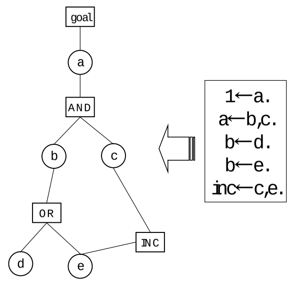

Fig. 1. Hypothetical Reasoning P: represented by a Network.

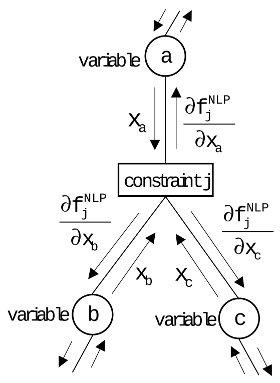
variable a
x_a
$\frac{\partial f_j^{NLP}}{\partial x_a}$
constraint j
$\frac{\partial f_j^{NLP}}{\partial x_b}$
$\frac{\partial f_j^{NLP}}{\partial x_c}$
x_b
x_c
variable b
variable c

Fig. 2. Message-passing.

Out of the many parallel computation techniques that deal with constrained optimization, we decided to use the augmented Lagrangian method, a method of multipliers that successively updates primal and dual variables. It is an excellent general purpose method for constrained optimization problems [Bertsekas and Tshitshiklis 89].

3.1 Augmented Lagrangian Method

We briefly introduce the augmented Lagrangian method. Consider the constrained optimization problem

$$\text{Minimize} \quad F(x) \tag{4}$$

$$\begin{aligned} \text{subject to} \quad & h_j(x) = 0 \quad (j \in C) \\ & x \in P, \end{aligned} \tag{5}$$

where C is a set of constraints, and P a nonempty polyhedral set. The Lagrangian function is

$$L(x) = F(x) + \sum_{j \in C} p_j h_j(x).$$

We consider two kinds of processors. One is a *Variable processor*, which is assigned to each prime variable x_i. The other is a *Constraint processor*, which treats the dual variable p_j. A Variable processor updates the prime variables so as to reduce cost of the Lagrangian function. Conversely, a Constraint processor updates the dual variables to increase the function. By the iteration of this process, the (optimal) solution is obtained.

To update dual variables, an important property is the strict convexity of the primal cost function $F(x)$, since it implies differentiability of the dual cost function $\inf_{x \in P} L(x)$. We add a quadratic term to the cost function, and instead of the original objective function (4), we consider the following objective function

$$F_c(x) = F(x) + \frac{c(t)}{2} \sum h_j(x)^2,$$

where t is the count of iterations and $c(t)$ is a positive scalar parameter or a nondecreasing function which is tuned experimentally[2]. The Augmented Lagrangian function is

$$L_c(x) = F_c(x) + \sum_{j \in C} p_j h_j(x).$$

[2] According to [Bertsekas and Tshitshiklis 89], practical experience has shown that it is best to start with a moderate value of c and then either keep c constant, or increase c by some factor (say, 2 to 10) with each minimization of the Augmented Lagrangian function.

The Augmented Lagrangian method consists of successive minimizations by the Variable processor of the form (":=" means "is defined by")

$$x_i := \arg \min_{x_i \in P_i} L_c(x) \tag{6}$$

followed by updates of the Constraint processor according to

$$p_j := p_j + c(t) h_j(x). \tag{7}$$

3.2 Application to Hypothetical Reasoning Problem

Now we are ready to apply the Augmented Lagrangian Method to hypothetical reasoning problems. Under the equality constraint (3), we minimize the objective function (1) as follows.

$$L(x) = \sum_{i \in N} w_i x_i + \sum_{j \in C} p_j h_j^{EQ}(x) + \frac{c(t)}{2} \sum_{j \in C} h_j^{EQ}(x)^2$$

The messages passed between Variable processor and Constraint processor are shown in Figure 3. From a Constraint processor j to a neighboring Variable processor i, the partial derivative

$$\frac{\partial L_j}{\partial x_i} = p_j \frac{\partial h_j^{EQ}(x)}{\partial x_i} + c(t) h_j^{EQ}(x) \frac{\partial h_j^{EQ}(x)}{\partial x_i} \tag{8}$$

is sent, where L_j consists of terms in $L(x)$ that are related to constraint j. The Variable processor sums up these values and adds its own weight as follows.

$$\frac{\partial L}{\partial x_i} = w_i + \sum_{j \in Neighbor(i)} \frac{\partial L_j}{\partial x_i}$$

If $\partial L / \partial x_i$ is positive, decrease x_i. If $\partial L / \partial x_i$ is negative, increase x_i. Thus the Variable processor updates its variable to make $\partial L / \partial x_i$ to be 0. This is what one Variable processor does in one iteration. As a result, L is minimized with respect to x_i.

On the other hand, a Constraint processor receives the messages and updates their variable according to (7). As a result, L increases. A Constraint processor based on the equality constraint (3) will be called *EQ processor*.

Similarly, we will call a Constraint processor based on the linear inequality constraint (2) an *LI processor*. A LI processor can be built in the similar way as a EQ processor, though the formulas of both the message and update are more complicated (for details, see [Bertsekas and Tshitshiklis 89]).

Fig. 4(a) shows the network constructed by EQ processors and Variable processors. Their interaction realizes the search to minimize (1) under the equality constraint (3). Fig. 4(b) is constructed by LI processors and Variable processors. The operation of this network corresponds to the simplex method, although they differ in constructing a search path and efficiency.

4 Relation to other Algorithms

Although a message from a EQ processor to a Variable processor represented by (8) is rather complicated, it does not have to be precise. It is already helpful if the message contributes to decrease the Lagrangian function by the Variable processors. In this section, first we ignore the second term of (8). In this way, the iteration becomes the breakout type algorithm that works in 0-1 space. Second, if we ignore the first term of (8), Gu's nonlinear optimization is obtained.

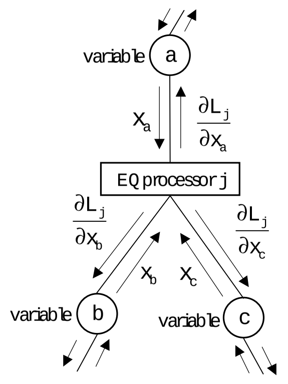
variable
a
x_a
$\frac{\partial L_j}{\partial x_a}$
EQ processor j
$\frac{\partial L_j}{\partial x_b}$
$\frac{\partial L_j}{\partial x_c}$
x_b
x_c
variable
b
variable
c

Fig. 3. Messages of EQ Processor.

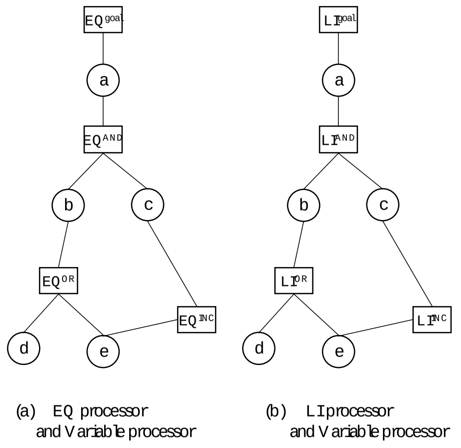
EQ goal
a
EQ AND
b
c
EQ OR
EQ INC
d
e
(a) EQ processor and Variable processor
LI goal
a
LI AND
b
c
LI OR
LI INC
d
e
(b) LI processor and Variable processor

Fig. 4. Constraint/Variable Processors.

4.1 Relation to the Breakout Type Algorithm

Assume that we ignore the second term of the formula (8).

– Variable processor i updates its value so that L is minimized. However, since L is linear with respect to x_i, the variable takes only the end point, which means 0 or 1 (or the current value).
– EQ processor updates its value according to (7). Because the prime variables take only 0 or 1, $h_j^{EQ}(x)$ is either 0 or 1. That is, if the neighboring variables violate the Horn clause, p_j is increased by $c(t)$.

In other words, a Variable processor takes only 0 or 1 and flips its value so that the sum of the weights of violated constraints is minimized. A EQ processor increases the weight of the constraint only when it is violated. Observe that this algorithm is of the same type as the breakout method. It is interesting to see that we started from continuous-space optimization and obtained an 0-1 space algorithm.

This algorithm is very efficient as will be shown in Section 6 with respect to both search time and quality of the solution.

4.2 Relation to Gu's Nonlinear Optimization Method

What will happen if we ignore the first term of the formula (8)? Dual variables are ignored, and the behavior of the EQ processors is only based on $h_j(x)$, i.e., to what extent the constraints are violated. This is equivalent to minimize

$$f = \sum w_i x_i + c(t) \sum (h_j^{EQ}(x))^2$$

under no constraints. The resulting algorithm is of a similar type as Gu's method, described in Sec 3.2. However, due to the lack of dual variables, the search can often get trapped in local minima. We have to add a local handler to escape from a local minimum as done in the SL method. The SL method focuses on the violation of a Horn clause when search is trapped in a local minimum. In order to satisfy the violated Horn clause, it applies a sophisticated mechanism to fix the variables to 0 or 1.

5 Least EQ method

As the search proceeds, the values of dual variables p assigned to EQ processors grow. It makes the Lagrangian function differ from the original objective function, and the minimal point of the Lagrangian function has an increased distance from the minimal point of the original objective function. Although this contributes to find a feasible solution, some devices are needed to find a near-optimal solution.

Recall that if we use LI processors, we get the real-number optimal solution (and sometimes the strict optimal 0-1 solution). LI processors are good at finding a low-cost solution. How can we incorporate these LI processors to EQ processors? In this section, we describe how to merge these two kinds of processors.

In the search procedure, a Variable processor i sums up the partial derivative $\partial L_j/\partial x_i$ from neighboring Constraint processors and updates its value. Therefore, we can let LI processors and EQ processors work together.

The following algorithm describes the collaboration of two processors.

1. Compute the continuous-value solution by LI processors. If the solution is 0-1 solution, terminate.
2. Else, install a EQ processor to the violated Horn clause.
3. For each processor, update its value. Iterate until the system converges. If it converges, go to 2. During the iteration, try to approximate the value of each variable to 0-1, and if a feasible solution (i.e., one satisfying all Horn clauses) is obtained, go to 4.
4. Try to temporally change each true element hypothesis to false without defeating the proof of the goal. If this succeeds, change the element hypothesis in question and its associated intermediate nodes to false; i.e., remove this element hypothesis from the solution hypothesis as a redundant one.

This algorithm tries to explore the search space created by LI processors, guided by EQ processors. In the second procedure, we employ this strategy: to one constraint whose $h_j^{EQ}(x)$ is the greatest (i.e., the most violating), we install a EQ processor. As will be shown in Section 6, this brings us a high-quality solution because we use a smaller number of EQ processors.

The features of this method are:

– Though we add a EQ processor at each convergence, we do not substitute 0/1 for any variables. There is no need to backtrack nor is the search space narrowed.
– When the iteration restarts after convergence, each processor uses the value resulting from the previous convergence. Therefore the search converges rather quickly after the first convergence. For the first convergence, we can use a more sophisticated method such as the simplex method.

This algorithm does not guarantee the strictly optimal 0-1 solution. However, as the number of EQ processor to be added is very small (less than few processors in the experiment of Section 6), it brings high-quality near-optimal solutions.

6 Experimental Results and Evaluation

We tested the performance of the methods that have been described in this paper. The system is implemented in C++ and runs on SGI Onyx workstations. The considered problems are randomly generated following these conditions: the number of body atoms in each Horn clause is 2–7, and the number of the occurrences of each atom in one sample knowledge set is at most 10. Two different types of problems are used: the first type has no inconsistency constraints (denoted as problem A), whereas the second type involves 18%–30% inconsistency constraints (denoted as problem B).

Table 1. Quality of Solutions (problem A)

	0/EQ	LP/EQ	SL	0/breakout	LP/breakout	LP/LI+EQ	LP/LI+leastEQ
Failed	0	0	0	0	0	0	0
Average of Costs	115.43%	105.74%	100%	98.88%	95.92%	93.85%	92.37%
Score to SL meth.	50-167-44	57-114-100	0-0-271	87-91-93	95-56-120	93-31-147	110-22-139

271 problem instances are used.
"Averages of Costs " shows the average cost percentage to the cost by SL method.
"Score to SL method" shows Win-Lose-Even against SL method.

Table 2. Quality of Solutions (problem B)

	0/EQ	LP/EQ	SL	0/breakout	LP/breakout	LP/LI+EQ	LP/LI+leastEQ
Failed	73	73	119	73	73	73	73
Average of Costs	119.33%	105.01%	100%	99.71%	98.49%	96.84%	95.88%
Score to SL meth.	21-89-42	29-55-68	0-0-271	41-41-70	42-30-80	45-23-84	46-22-84

271 problem instances are used.

In the current stage, the roughly categorized two types of problem instances are used. But needless to say, it is desirable to define the problem instances more clearly according to some criterion. As briefly described in the last section, detailed analyses have to be defered to future work. Note that the following results are still preliminary but sufficient to obtain an estimation of the performance of our algorithms. The discussion mainly focuses on computational orders and quality of solutions.

The following methods have been tested.

LP/EQ Use the continuous-value solution as an initial point. Install EQ processors to all the constraints.

0/EQ Set all the variables to 0 as the initial point. Install EQ processors to all the constraints.

SL method Use the continuous-value solution as an initial point. Solve unconstrainted nonlinear optimization problem using local handler.

LP/breakout type Use the continuous-value solution as the initial point. Install EQ processors (breakout type) to all the constraints. Note that the variables take continuous values only at the first iteration. After that, the search proceeds in the 0-1 space.

0/breakout type Set all the variables to 0 as the initial point. Install EQ processors (breakout type) to all the constraints.

LP/LI+EQ Use the continuous-value solution as an initial point. Install both LI and EQ processors to all the constraints.

LP/LI+leastEQ Use the continuous-value solution as an initial point. Install LI processors to all the constraints. If the system converges, add one EQ processor.

The respective inference times are summarized in Figures 5 and 6. Tables 1 and 2 contain details about the solution quality of the considered methods. Although the inference time of every method is polynomial with respect to the number of nodes, the simplex method turns out to be the slowest. Recall the the simplex method is used to get an initial search point. Its running time is included in the **LP/** series. The computational time of the simplex method is on the order of $n^{2.2}$, where n is the number of nodes. After getting the continuous-value solution, the SL method finds a 0-1 solution in approximately $n^{1.8}$, the breakout-type in approximately $n^{1.25}$, and LI+EQ and LI+leastEQ in almost n^1. Therefore, using the continuous-value solution as an initial point, the computational time of the simplex method gets dominant if applied to large-scale problems.

If we want a method whose computational time is on an order lower than $n^{2.2}$, we cannot use the continuous-value solution. Among the methods that do not use the simplex method, i.e., the **0 /** series, the best method is the **0/breakout type**. It achieves computational time on the order of $n^{1.25}$ and the quality of its solutions is comparable to the SL method.

Let us assume that we want good-quality solutions and are allowed to use the simplex method, i.e., solution quality is our main concern. In this case, our results show that the leastEQ method (**LP/LI+leastEQ**) should be the preferred method. The result can also

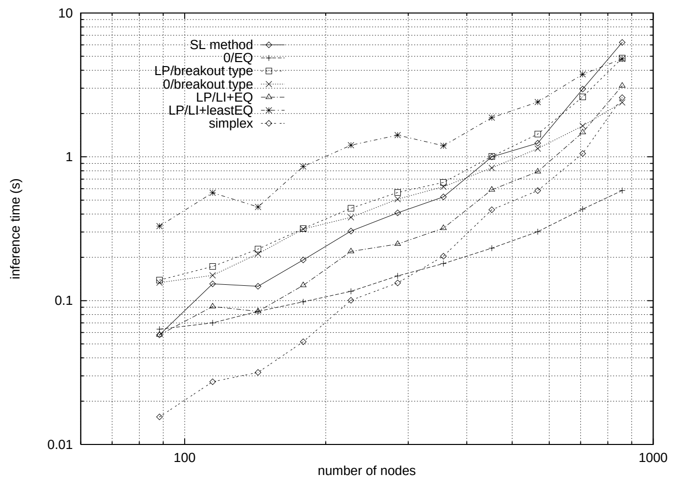

Fig. 5. Inference Time (problem A)

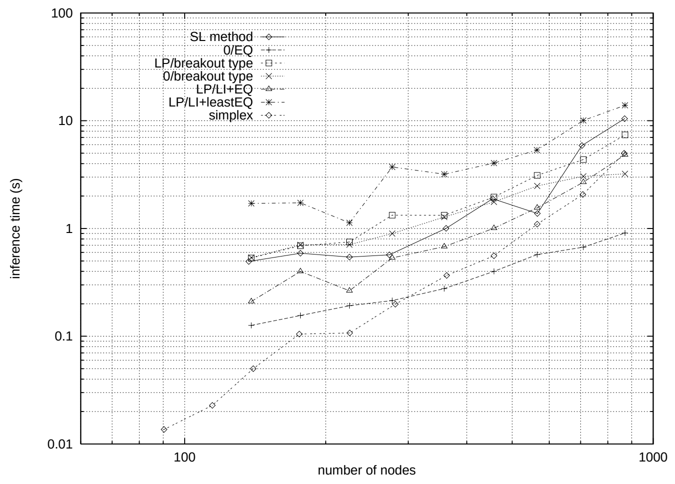

Fig. 6. Inference Time (problem B)

be justified theoretically. We can show that the search is the least likely to gain distance from the real-number optimal solution.

To summarize, we have described two competitive methods: one is breakout type (without simplex method), the other is the leastEQ method. They have tradeoffs: the breakout type trades solution quality for speed and the leastEQ method is slightly slowered but yields high-quality solutions.

7 Conclusions and Future Work

In this paper, in order to grasp the search of primal and dual variables intuitively, we employed a parallel processor approach. The relation between several algorithms is discussed and two competitive methods for hypothetical reasoning are described. The first one is similar to the method of constraint-weighting such as the breakout algorithm. It is shown to be also effective for hypothetical reasoning problems. The second one is an original method based on linear programming, where equality constraint are added one by one. A notable feature of this method is its good solution quality.

In this paper we didn't show detailed results relating to runtime distributions. This is partly because there seems to be no measure to scale the difficulty of the problem classes for hypothetical reasoning problems. But if the analyses of SAT problems are introduced to hypothetical reasoning problems as well, it may be possible to scale the difficulty. For example, we can exploit an early analysis of SAT problems, that reveals the probability for a problem to have feasible solutions [Franco and Paull 83], to hypothetical reasoning problems. Our future work is thus to experiment in detail our methods for various classes of problems and hopefully to show a similar phenomenon to phase transitions [Gomes and Selman 97] in hypothetical reasoning problems. By applying randomization, we may enhance the performance of our methods, as done by [Gomes *et al.* 00].

Another direction of our work is to make our methods anytime algorithms. For example, [Yokoo and Hirayama 96] is an application of the breakout method for distributed constraint satisfaction problems, which has features of an anytime algorithm. This improvement may increase the applicability of our methods to real world problems.

References

[Bertsekas and Tshitshiklis 89] D. P. Bertsekas, J. N. Tsitsiklis, "Parallel and Distributed Computation", Prentice-Hall (1989)

[Franco and Paull 83] J. Franco, M. Paull: Probabilistic Analysis of the Davis Putnam Procedure for Solving the Satisfiability Problem, Discrete Applied Mathematics, 5, pp.77–87 (1983)

[Gomes and Selman 97] C. Gomes, B. Selman: Problem Structure in the Presence of Perturbations, Proc. AAAI-97, pp.221–227 (1997)

[Gomes *et al.* 00] C. Gomes, B. Selman, N. Crato: Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems, J. of Automated Reasoning, Vol.24, pp.67–100 (2000)

[Gu 94] J. Gu: Global Optimization for Satisfiability Problem, IEEE Trans. on Knowledge and Data Engineering, Vol.6, No.3, pp.361–381 (1994)

[Ishizuka and Matsuo 98] M. Ishizuka, Y. Matsuo: SL Method for Computing a Near-optimal Solution using Linear and Non-linear Programming in Cost-based Hypothetical Reasoning, Proc. PRICAI'98, pp.611–625 (1998)

[Minton *et al.* 92] S. Minton, M. D. Johnston, A. B. Philips, P. Laird: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence Vol.58, pp161–205 (1992)

[Morris 93] P. Morris : The Breakout Method for Escaping from Local Minima, Proc. AAAI-93, pp.40–45 (1993)

[Ohsawa and Ishizuka 97] Y. Ohsawa, M. Ishizuka : Networked Bubble Propagation: A Polynomial-time Hypothetical Reasoning Method for Computing Near-optimal Solutions, Artificial Intelligence, Vol.91, pp.131–154 (1997)

[Poole 88] D. Poole: A Logical Framework for Default Reasoning, Artificial Intelligence, Vol.36, pp.27–47 (1988)

[Santos 94] E. Santos,Jr.: A Linear Constraint Satisfact ion Approach to Cost-based Abduction, Artificial Intelligence, Vol.65, pp.1–27 (1994)

[Selman and Kautz 93] B. Selman, H. Kautz: Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems, proc. IJCAI-93, pp.290–295 (1993)

[Thornton and Sattar 99] J. Thornton, A. Sattar: On the Behavior and Applicatoin of Constraint Weighting, Principles and Practice of Constraint Programming - CP'99 Lecture Notes in CS 1713, pp.446–460, Springer (1999)

[Voudouris 95] C. Voudouris, E. P. K. Tang: Guided Local Search, Technical Report CSM-247, University of Essex, UK (1995)

[Wah and Shang 97] B. W. Wah, Y. Shang: Discrete Lagrangian-Based Search for Solving MAX-SAT Problems, Proc. IJCAI-97, pp.378–383 (1997)

[Yokoo and Hirayama 96] M. Yokoo, K. Hirayama: Distributed Breakout Algorithm for Solving Distributed Constraint Satisfaction Problems, Proc. the Second International Conference on Multiagent Systems (ICMAS-96), pp.401–408 (1996)