A Quick Adaptation Method for Constraint Satisfaction in a Real-time Environment

Hiromitsu Hattori and Takayuki Ito

Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya, Aichi, 466-8555 JAPAN

Summary
Multi-agent problem solving in a real-time environment is one of the hardest and most fascinating research area. In this paper, we propose a method for quick adaptation to changes in a problem in a real-time environment as an extension of an Asynchronous Weak-commitment (AWC) search algorithm. The basic idea of our proposing method is to limit the scope of search. Variables which are affected by changes could be modified their previously assigned values to adapt to the changes. For the adaptation to changes in a problem, we sacrifice the optimality of a solution. If an agent can increase its utility, it can simply modify its values. In the process, an agent emphasizes the local optimality without considering the global optimality. In a real-time environment, it is not practically important to globally optimize a solution. The quick adaptation to changes can be useful for such real-time problem.

Key words:
Distributed Constraint Satisfaction Problem, Real-time problem, Multi-agent Systems

Introduction

There are many researches which try to apply multi-agent systems for various application domains. Multi-agent problem solving in a real-time environment is one of the most fascinating research issue [2]. In such environment, a problem could be dynamically changed over time, and then the property of a problem might be changed. Although agents should adapt to changes in a problem, there is a trade-off between the efficiency of the computation time and the quality of a solution. We have focused on the efficiency of the calculation to adapt to changes in a problem because in the real-time environment, where a problem might be changed in a shorter period, agents should respond to changes at the cost of the solution quality.

Constraint Satisfaction Problem (CSP) is one of the promising framework for achieving multi-agent problem solving. It is not, however, efficient to deal with many real-time problems with dynamic changes because in the classical CSP, it is assumed that problems are static. Dynamic CSP is a framework in order to deal with such intractable problem. Many existing researches, however, has focused on solution quality which is in the light of the stability, consistency, and so on. The efficiency of problem solving has not been focused on. In this paper, we describe a real-time problem with constant changes and an adaptation method which allows us to obtain a solution quickly.

We are assuming problems such as a disaster rescue problem based on multi-agent systems. For example, in a disaster rescue problem [5], agents are firstly assigned tasks, and resources. Then, they execute assigned tasks using the resources. While agents are working, there might be suddenly kinds of changes, *e.g.,* coming of new agents, communication blackout, detection of new sufferers, and outbreak of fire, and so on. For these changes, agents should respond to such kinds of changes. However, it is not necessarily required that all agents change their ongoing tasks to respond to them. Additionally, in a real-time problem that agents must constantly recalculate their tasks, an exact/optimal solution at each moment is not required. This is because that the effectiveness of a solution in a real-time problem might be reduced due to changes. Therefore, in this paper, we propose a method to achieve quick adaptation to changes in a real-time problem.

In this paper, we represent a real-time problem based on a Dynamic Distributed CSP. Changes in a problem is described as the modification of a set of constraints and variables, *i.e.,* addition/deletion of constraints and variables. For quick adaptation to changes in a real-time problem, we extend the Asynchronous Weak-commitment search algorithm (AWC) [8]. The basic idea of our proposing method is to limit the scope of search. When some changes occur, a previous solution could be locally modified. Variables which are affected by changes could be modified their previously assigned values to adapt to the changes. According to our method, the number of variables which is required to modify each value could be decreased, and then agents affected by changes could quickly respond to them. For the adaptation to changes in a problem, we sacrifice the optimality of a solution. During a search process in our method, agents consider only each own utility. If an agent can increase its utility, it can simply modify its values. If not, the agent communicates

Manuscript received July 5, 2006.
Manuscript reviced July 25, 2006.

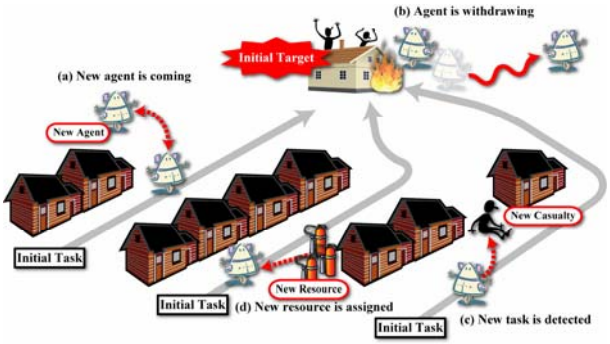

Fig.1. An Image of the Real-time Problem

with neighbours to achieve an appropriate modification. In the process, an agent emphasizes the local optimality, thus the global optimality is not guaranteed. However, it is not important to globally optimize a solution since we assume an environment where a problem could be constantly altered. On the contrary, the quick adaptation to changes can be useful for such real-time problem.

The structure of the rest of this paper is as follows. In the next section, we describe a real-time problem on which we have focused. In Section 3, we show the definition of Constraint Satisfaction Problem as preliminaries. In Section 4, we describe the description of a real-time problem for our proposing method. Then, in Section 5, we show a quick adaptation method for changes in a real-time problem. In Section 6, we discuss some related researches.

2. Real-time Problem

We focus on the problem solving in a real-time environment in this paper. Figure 1 shows an example of a problem. In this example, several agents try to extinguish the fire and rescue sufferers. There are also some incidents which causes changes. The changes in a problem is mainly caused by incidents such as (1) Alteration of the number of agents, (2) Occurrence of new tasks, (3) Assignment of new resources. In a rescue problem, as an incident (1), new rescue agents could come in the process of rescue activities. Meanwhile, some agents withdraw from the activities due to fatigue, a sort of accidents, and so on. In this case, agents should determine how to cooperate with new comer and how to cover the lack up agents. As shown in Figure 1 (a), an agent which perceives new agent is coming should determine how to cooperate with the new one. In other cases, an agent might continue to execute assigned tasks and does not care the new agent. Figure 1 (b) shows a case that an agent which has worked together could withdraw in the process of a task. In this case, the remaining agent might search for other potential agents for cooperative work. As an incident (2), new tasks could suddenly emerge during an activities. When an agent detects new sufferers, outbreak of fire, and other kinds accidents, it must determine how it should do considering such unexpected tasks. As shown in Figure 1 (c), an agent finds unexpected casualty. In this case, the agent must determine whether it continues to execute its initial task or try to rescue the casualty instead of the initial task. As an incident (3), additional resources for activities could be newly assigned and sometimes existing resources could be exhausted. This case is similar with the case for the type of incident (1). As shown in Figure 1 (d), if an agent can get much more extinguisher, it might not need to work with others.

Due to kinds of unexpected incidents, an agent might be occasionally required to modify how it work considering such incidents. However, all of agents might not be necessarily required to change ongoing tasks. For example, in Figure 1, assuming that an agent in (c) changes its task into the rescue of new casualty. Generally, there is the possibility that the optimal solution could be obtained as a result of the re-calculation by all agents. However, if the advantage of such optimal solution is not large enough, such optimization is practically waste of time.

Because changes in a problem, we are considering in this paper, are unexpected, it is impossible to initially take such changes into account at problem modeling phase, and agents cannot anticipate the effect on the solution quality. For the problem solving in such uncertain environment, it is not important to obtain an optimal solution to the problem at each moment. This is because an optimal solution to a problem might be obsolete one at the next moment. The more desired property for the problem solving in a real-time environment is to quickly respond to the changes. For example in Figure 1, there is a case that new agent is coming after an agent in (c) stops achieving the initial target. In this case, it is a waste of time to try to obtain an optimal solution again among all agents considering the absence of an agent in (c) because in this example new agent, which can recover the absence, is coming at next step. Therefore, it might be reasonable for agents to execute each task continuously. Accordingly, in some cases, the global optimization at each moment might be meaningless for a real-time problem due to uncertainty in such problem.

Finally, we show the definition of a problem in a real-time environment in this paper:

Definition 1 (The Real-time Problem) *The real-time problem is constantly changing due to some kind of unexpected incidents. Since the property of a problem could be altered due to the changes, the solution in a moment could be obsolete.*

3. Distributed Constraint Satisfaction Problem

A classical Constraint Satisfaction Problem (CSP) is denoted as $P = (X, D, C)$. Each term represents the following components:

- a set of variables $X = \{x_1, ..., x_n\}$
- a set of finite domains for each variables $D = \{d_1, ..., d_n\}$
- a set of constraints $C = \{c_1, ..., c_n\}$

A solution to the CSP, A, is an assignment of values to all of the variables which can satisfy all constraints. A classical CSP is static. The problem is fixed in advance and does not change afterwards.

A Distributed CSP (DCSP) is the problem that each agent is assigned one or more variables and constraints [9]. Each agent is assigned a subproblem. The goal of each agent in a DCSP is to assign values which can satisfy all constraints. The goal of each agent is dependent on other agents' goal since some variables included in each problem are related by *inter-agent constraints*. Therefore, agents must communicate with others to fix variable values. When all agents assign values of all own variables and such values satisfy all local constraints, a DCSP is solved. The Asynchronous Weak-commitment search algorithm (AWC) is one of the algorithm to solve a DCSP. In AWC, high priority agents can keep their values and the priority is dynamically changed. Agents communicate each other by using *ok?* and *nogood* messages. When an agent receives an *ok?* message, if an agent cannot find a value to its variable that is consistent with values of higher priority agents, the agent creates and sends a *nogood*. Actually, *nogood* is a constraint which can avoid an assignment which cannot satisfy all constraints. AWC is guaranteed to be complete by recording all nogoods.

Dynamic CSP is an extension of a classical CSP [1]. If there are kinds of changes to a problem, *i.e.,* addition/remove of constraints and variables, the problem is altered. The problems with dynamic changes can be defined as a sequence of static CSPs. In the sequence, each problem is generated from the previous one. Letting P_t be a static CSP at time step t, we can then represent the a Dynamic CSP DP as follows:

$$DP = \{P_0, P_1, ..., P_t, P_{t+1}, ...\}$$

where P_{t+1} is a problem generated from the previous problem P_t. More specifically, there is a function F which define how a problem is altered. The function F can generate P_{t+1} from P_t, *i.e.,* $P_{t+1} = F(P_t)$. F can be defined as a function to add/remove a subset of variables and constraints in the previous problem.

We will describe a real-time problem as a Dynamic DCSP in this paper. As mentioned above, in a DCSP, each agent is assigned variables and constraints which are a part of a problem, then a problem at each moment can be solved based on AWC. However, assigned variables and constraints could be dynamically changed. Assuming that one variable represents an agent, for example, when new agent is added to a problem, some existing agents related to new one via constraints could be much constrained. On the other hand, when some agents are removed from a problem, related agents are relaxed because of the decrease of the number of constraints. We can solve a Dynamic DCSP by executing AWC for each problem, but it is time consuming.

4. Problem Description

We show the formal representation of a real-time problem considered in this paper. First, a real-time problem itself is described as $RP = \{P_0, P_1, ..., P_t, P_{t+1}, ...\}$. P_t, a problem at time step t, is denoted as $P_t = (X_t, D_t, C_t)$, and P_t is a DCSP. For simplicity, we assume that one variable in a problem represents an agent. RP is a problem which could be constantly changed over time. Therefore, it might be difficult to achieve a solution to each DCSP which can satisfy all constraints. Because a problem at each time is generally over-constrained, some of agents should accept that some constraints are not satisfied with the solution.

As we mentioned in Section 1, we do not consider all variables and constraints during the search process, but the limited number of them. That is, in fact, a subproblem is solved in the search process. The definition of a subproblem at time step t, S_t, is as follows:

$$S_t = (SX_t, SD_t, SC_t) \quad (SX_t \subseteq X_t, SD_t \subseteq D_t, SC_t \subseteq C_t)$$

where SX_t, SD_t, SC_t represent a subset of X_t, D_t, C_t, respectively. An agent which is subject to the search at t is $X_t \in SX_t (i = 0, ..., |SX_t|)$ and a set of constraints related to X_t is $C_t = \{c_0, ..., c_n\} \subseteq sc_t$. An agent can evaluate its utility at a moment. The definition of the utility is as follows:

$$u(c_j) = \begin{cases} f(t, c_j) + w_{c_j} & : if\ c_j\ is\ satisfied\ by\ A_t \\ 0 & : otherwise \end{cases}$$

That is, when a constraint c_j is satisfied with a certain assignment A_t, which is a solution to a problem at t, an agent can obtain the utility. The utility is defined as the combination of two factors, *i.e.,* the time-depend value and the importance of a constraint. First, we suppose that the value of a constraint depends on the time. For example,

```
 1: procedure search(Starter, Actives, Th)
 2:   broadcast "activate(Starter, Th)" to agents in Actives
 3:   wait until all active neighbours return "ready" message
 4:   search a solution using AWC considering utility condition

 1: procedure extend scope(Extender)
 2:   Consts := {c|a constraint related to starter}
 3:   Satisfy := {d|Extender 's value which can satisfy a
      constraint in Consts}
 4:   if Satisfy = ∅ then
 5:     return nil
 6:   else if Th = 0
 7:     return nil
 8:   else
 9:     D_i := Satisfy
10:    Affected := {c|a previously satisfied constraint which
         are dissatisfied with a value in D_i}
11:    if Affected = ∅ then
12:      return nil
13:    else
14:      C_i := C_i ∪ Affected
15:      New actives := a set of neighbours related to C_i
16:    end if
17:  end if
18:  send "reset" message to all active neighbours
19:  Th := Th - 1
20:  search(Extender , New actives, Th)

 1: when i received (activate(Sender ,Th))
 2:   store Sender as a Starter
 3:   store Th
 4:   broadcast "know(sender)" to all neighbours
 5:   wait until all neighbours reply
 6:   C_i := agent i's constraint set
 7:   for each neighbour j
 8:     if j's reply = "replied(j, ∅)" then
 9:       C_j := constraints related to j
10:      C_i := C_i – C_j ∪ a constraint "j takes its current value"
11:    end if
12:  end for
13:  send "ready(i)" to Sender
14: end

 1: when i received (know(Sender))
 2:   if i is activated by Sender then
 3:     send back "replied(i, ∅)" to Sender
 4:   else
 5:     send back "replied(i,i's current value)" to Sender
 6:   end if
 7: end
```

Fig. 2. Basic Algorithm

if it is known that an agent X_i withdraws within a minute, a constraint on the cooperation with X_i could not deserve to be satisfied. Additionally, a constraint has the importance of itself, which is regardless of the time. Accordingly, we define a function to set the utility as summation of the value considering the time, $f(t, c_j)$, and the importance of a constraint, w_{cj}. Eventually, the utility of an agent X_i can be described as the following formula:

$$U^i = \sum_j^{|C^i|} u(c_j)$$

An agent tries to increase its utility during the search process. However, although we will show in the next section, we do not consider a problem as an utility optimization problem.

In this paper, we assume that a change occurs on one agent. To put it concretely, one new agent comes/withdraws or constraints related to one agent are added/removed. Accordingly, at each moment, only one agent has motivation to re-search a solution.

5. The Adaptation Method for a Real-time Problem

In this section, we describe a quick adaptation method for changes in a real-time problem. Our proposing method is basically based on an existing search algorithm, Asynchronous Weak-commitment search algorithm (AWC). Because, as we mentioned above, a problem at a moment would be over-constrained, the method accepts that some constraints are not satisfied. In order to obtain a solution, AWC search processes are iteratively executed. During the execution of AWC search, the number of variables and constraints which are considered in the AWC search process is limited within much less number of them. Therefore, the computation time for each search would be small.

Figure 2 shows an algorithm for the quick adaptation. In the following, we present a sketch of each step of our algorithm.

Step 1: An agent, *Starter*, which is added to a problem or modified its constraint set runs *search* procedure. Here, *Actives* is a set of neighbours of *Starter*. *Th* is a threshold

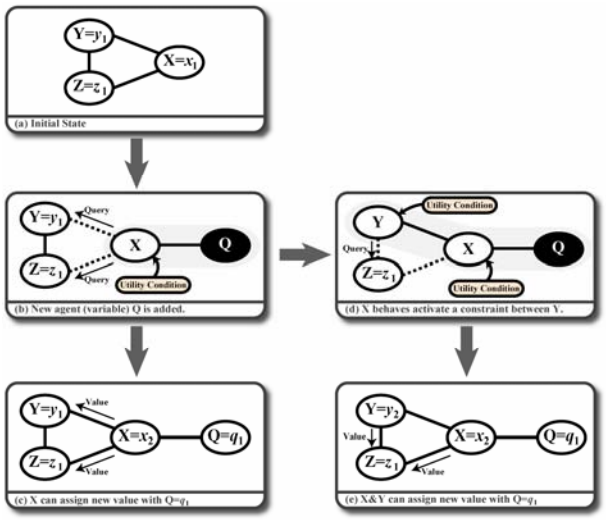

Fig. 3. An Example of the re-calculation

that defines the limit of extension of the scope of search. If it is possible to extend the scope, *i.e.,* $0 \leq Th$, *Starter* broadcasts *activate* message to activate neighbours (*search* procedure: Line 2 in Figure 2). A set of active agents corresponds to the scope of search.

Step 2: An agent which receives *active* message, here we call *receiver*, broadcasts *know* message (*search* procedure: Line 4 in Figure 2) to figure out which agents are active.

Step 3: An agent which receives *know* message replies. If an agent also receives *activate* message, it is active, then sends back an empty message. On the other hand, if an agent does not receive the message, it sends back its own assigned value.

Step 4: *Receiver* temporarily eliminates constraints related to inactive agents. Additionally, *receiver* generates temporal hard constraints to keep existing value of inactive agents. Thus, although active agents do not communicate with inactive ones through the search process, the value of inactive agents can be considered.

Step 5: *Starter* runs AWC search algorithm with only active agents. Supposing that the utility of an agent x_i at time step t is described as $U_{(i,t)}$, during the search process, x_i must satisfy the following condition:

$$U_{(i,t-1)} \leq U_{(i,t)}$$

When an agent x_i does not satisfy this condition, it must send *nogood* message to its active neighbours even if it satisfies several constraints. That is, x_i must refuse an assignment which reduces its utility.

Step 6: When an agent, *Extender*, cannot find a value which can increase its utility, it can try to extend the scope if the following conditions are satisfied:

- There are potential values which can satisfy constraints related to *Starter*.
- *Th* is more than 0

For the extension, *Extender* temporarily replaces its domain with a set of value which can satisfy constraints related to *Starter*. Then, *Extender* checks whether the previously satisfied constraints are satisfied or not with the replaced domain. If there are constraints which are not satisfied with the domain, *Extender* considers an agent related to such constraints as an active one. After that, *Extender* runs *search* procedure with reset of existing *nogood* used in AWC. At this point, *Extender* becomes new *Starter*.

When an algorithm successfully finds a solution or returns *nil*, the search process is completed. If the algorithm fails to find a solution, all existing agent are assigned their previous values again.

Figure 3 shows an example of an execution based on the proposed algorithm. In this example, domains of three variables, *X*, *Y*, and *Z* are $\{x_1, x_2\}$, $\{y_1, y_2\}$, $\{z_1, z_2\}$, respectively. In the initial state, there are only three variables *X*, *Y*, and *Z*. Each of them is assigned x_1, y_1, and z_1 (Figure 3 (a)). New variable *Q* is added to *X* as shown in Figure 3 (b). As a *Starter*, *Q* sends *activate* message, and then *X* sends *know* message to *Y* and *Z*. Because both *Y* and *Z* do not receive *know* message from *Q*, the scope of search is only *X* and *Q*. Therefore, at this step, a sub-problem is $S_t = (\{Z,Q\}, \{D_{Xt}, D_{Qt}\}, C_{Xt} \cup C_{Qt})$. If *X* can assign another value x_2 which can satisfy all constraints with $Y = y_1$ and $Z = z_1$, *X* can obviously satisfy the condition on the utility, namely its utility is increased. In that case, a solution is as Figure 3 (c). If *X* cannot find a solution, it tries to extend the scope. As well as *Q*, *X* sends *activate* message, then *Y* fixes its active neighbour. As shown in Figure 3 (d), the scope consists of *X*, *Q*, and *Y*. Here, we assume that only $X = x_2$ can satisfy a constraint between *Q*, but cannot satisfy a constraint between *Y* with $Y = y_1$. In this case, a sub-problem is $S_{t+1} = (\{X, Q, S_t = (\{x_2, Q\}, \{D_{Xt}, D_{Qt}, C_{Xt} \cup C_{Qt}\}), \{x_1, D_{Qt}, D_{Yt}\}, C_{Xt+1} \cup C_{Qt+1} \cup C_{Yt+1})$. Figure 3 (e) shows a case which can successfully find a solution.

6. Related Work

There has been some researches on Multi-agent problem solving considering dynamic changes in a problem.

Verfaillie and Schiex [6] proposed a method of using a previous solution to produce a new solution. Wallace and Freuder [7] proposed a method that assigns a penalty for

constraint violation. Miguel and Shen [3] proposed fuzzy *rr*DFCSP, which can represent an over-constrained, dynamic problem. These researches focused on the solution stability, but did not focus on the efficiency of the computation in a real-time environment.

Modi et. al. proposed a constraint satisfaction method for a dynamic distributed constraint optimization problem [4]. Their work is close in spirit to our approach. However, they focused on how to optimize a solution, but we have focused on how to obtain a solution efficiently.

7. Conclusion

In this paper, we first described a real-time problem as a Dynamic Distributed CSP because classical CSP assumes static problems. Then, we extended an existing Asynchronous Weak-commitment search algorithm for quick adaptation to changes in a real-time problem. The main idea of our proposing method is to limit the scope of search at each moment. During the search process, each agent determines whether there is the possibility of increasing its own utility. Because the number of variables and constraints would be much smaller, the computation time could be reduced. Thus, according to our method, agents can quickly respond to changes.

Our future work will include the sophistication of the method to obtain better quality solution.

References

[1] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. *Proc. of AAAI-88*, pages 37–42, 1988.

[2] H. Hattori, T. Shintani, A. Isomura, Ito. T., and T. Ozono. Stable solutions dealing with dynamics in scheduling based on dynamic constraint satisfaction problems. *Proc. of PRICAI-04*, pages 989–990, 2004.

[3] I. Miguel and Q. Shen. Fuzzy *rr*dfcsp and planning. *Artificial Intelligence Journal*, 148(1-2):11–52, 2003.

[4] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. *Proc. AAMAS-2003*, pages 161–168, 1998.

[5] S. Tadokoro, H. Kitano, T. Takahashi, I. Noda, H. Matsubara, A. Shinjoh, T. Koto, I. Takeuchi, H. Takahashi, F. Matsuo, M. Hatayama, J. Nobe, and S. Shimada. The robocup-rescue project: A robotic approach to the disaster mitigation problem. *Proc. of ICRA-2000*, pages 4089–, 2000.

[6] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. *Proc. of AAAI-94*, pages 307–312, 1994.

[7] R. J. Wallace and E. C. Freuder. Stable solutions for dynamic constraint satisfaction problems. *Proc. of CP-98*, pages 447–461, 1998.

[8] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.

[9] M. Yokoo. Distributed Constraint Satisfaction. Springer, 2001.

Hiromitsu Hattori received the B.E., M.E., and Doctor of Engineering from Nagoya Institute of Technology in 1999, 2001, 2004, respectively. He has been a research fellow of the Japan Society for the Promotion of Science (JSPS) from 2004. From 2004 to 2005, he was a visiting researcher at University of Liverpool. From 2006, he has been a visiting researcher at Massachusetts Institute of Technology. His main research interests include multi-agent systems, agent argumentation, agent-mediated electronic commerce, and intelligent group decision support.

Takayuki Ito received the B.E., M.E, and Doctor of Engineering from the Nagoya Institute of Technology in 1995, 1997, and 2000, respectively. From 1999 to 2001, he was a research fellow of the Japan Society for the Promotion of Science (JSPS). From 2000 to 2001, he was a visiting researcher at University of Southern California. From 2001 to 2003, he was an associate professor of Japan Advanced Institute of Science and Technology. He joined Nagoya Institute of Technology as an associate professor of Graduate School of Engineering in 2003. He is a Founder, a Senior Vice President, Chief Operating Officer of Wisdom Web Co., Ltd. from 2004. From 2005 to 2006, he is a visiting researcher at Harvard University and Massachusetts Institute of Technology. His main research interests include multi-agent systems, intelligent agents, group decision support systems, and agent-mediated electronic commerce.