APPLYING DISTRIBUTED CONSTRAINT OPTIMIZATION METHOD TO DYNAMIC PROBLEM

Toshihiro Matsui and Hiroshi Matsuo
Department of Electrical and Computer Engineering
Nagoya Institute of Technology
Gokiso-cho,Showa-ku,NAGOYA,466-8555,JAPAN
email: {tmatsui@mars.elcom.,matsuo@}nitech.ac.jp

ABSTRACT
A framework which applies distributed constraint optimization method using depth first search tree to dynamic problem is proposed. The proposed framework is considered as a basic model of multi agent system. The agents communicate with each other to solve the problem. Constraint network for the problem is built. Trees, which are similar to depth first search tree for the constraint network, are built in a bottom-up manner. The problem is solved by distributed constraint optimization algorithm using the trees. Each distributed processing is executed asynchronously and the framework follows dynamic problem.

KEY WORDS
Multi Agent Systems, Dynamic Problem, Distributed Search, Constraint Optimization, Self Organization

1 Introduction

Distributed constraint optimization problem (DCOP) [1][2][3] is an extended distributed constraint satisfaction problem (DCSP)[4]. DCOP is considered as a basic distributed cooperation problem and it is an important area of study of multi agent system. Distributed search algorithms, which use depth first search (DFS) tree of constraint network, were proposed for DCSP and DCOP[5][6]. In the algorithms, agents are placed in DFS tree. There are no constraints between sub-trees of DFS tree. The parallelism of sub-trees makes effectiveness in search processing.

Basic distributed algorithms for construction of DFS tree [9] emulate sequential depth first search. On the other hand, the search algorithms, which uses DFS tree are applicable to any tree which has no constraints between subtrees, even if the tree is not exact DFS tree of constraint network. An idea to build the tree in a bottom up manner is shown in [5]. The pseudo DFS tree is considered as a spanning tree of constraint network, and it is also important for self organization of agents.

Dynamic constraint satisfaction problem[7][8] is an extended CSP. Dynamic CSP is formalized as a sequence of CSPs which have different sets of constraints, and the problems are solved in order. Generally, distributed environments including multi agent system are dynamic. Therefore the application of DCSP and DCOP to dynamic environment is important for practical purposes. For this reason, not only search of solution but also construction of problem and determination of solution should be realized by distributed algorithms. However, in most study of DCSP or DCOP, it is assumed that snapshot or global terminate detection algorithms are available for pre/post-processing of the search.

In this paper, we present a dynamic framework based on Adopt algorithm[6]. Our most important aim is how to implement the algorithm for dynamic distributed system. Constraint network for the problem is built. Trees which are similar to depth first search tree for the constraint network are built in a bottom-up manner. The problem is solved by distributed constraint optimization algorithm using the trees. Each distributed processing is executed asynchronously and the framework follows dynamic problem.

2 Dynamic distributed constraint optimization problem

Dynamic CSP is formalized as a sequence of CSPs which have different set of constraints, and the problems are solved in order. Dynamic distributed constraint satisfaction/optimization problem (DyDCSP/DyDCOP) is an extended DCSP/DCOP. Variables and constraints of the problem are distributed into agents. The problem is solved by distributed algorithm. In this paper, an instance of DyDCOP is considered.

Let A denote set of all nodes (agents). Each node $i \in A$ has a variable x_i. The variable x_i takes a value d_i in its domain D_i. Only node i is able to determine its value of x_i. In the following, we may use x_i instead of node i for the sake of simplicity.

Let P denote a sequence of problems. Let $P = \{p_0, \cdots, p_s, \cdots\}$, $p_s = (C_s, F_s)$. C_s is set of all constraints. F_s is set of cost function related to the constraint. For current problem p_s, a binary constraint $c_{i,j} \in C_s$ relates x_i to x_j. Cost of the assignment $\{(x_i, d_i), (x_j, d_j)\}$ is evaluated using $f_{i,j} \in F_s$. Global cost of the problem is conjunction (sum) of $f_{i,j}$ for all $c_{i,j}$. Optimal solution is the whole assignment which minimizes global cost. Current problem p_s changes into next problem p_{s+1} according to arbitrary schedule. The solution must be obtained for each problem before the problem changes.

Nodes have globally consistent knowledge of name and domain of variables (nodes). However, each constraint is observed by two nodes or only one node related to the

constraint. Therefore each node must obtain knowledge of constraints related to it before the problem is solved.

Each node has FIFO communication links to other nodes. We assume that nodes and communication links does not have any failure.

3 Construction of constraint network

Node i uses logical clock in order to identify current state of nodes. Let clk_i denote current logical time of the clock. When known constraints or state of nodes are changed, node i increments its logical time. Let Sts_i denote description of current state of node i. Sts_i includes information about constraints and state of nodes related by the constraints. Components of Sts_i are as follows.

- C_i^{out}, F_i^{out}: set of observed constraints and set of observed cost functions.
- C_i^{in}, F_i^{in}: set of received constraints and set of received cost functions.
- Deg_i^{in}: set of received degrees.
- Clk_i^{in}: set of received logical times.
- C_i, F_i: set of constraints and cost functions.
- Nbr_i: set of neighborhood nodes.

For the sake of simplicity, we identify each element of set using additional characters.

C_i^{out} and F_i^{out} are set of constraint and set of cost function which are observed by node i. If $c_{i,j} \in C_i^{out}$ or $f_{i,j} \in F_i^{out}$ is changed, node i must notify related node j of the change. Possible cases of changes are as follows. (1) Addition of new constraints. (2) Change of constraints. (3) Disappearance of constraints.

C_i^{in} and F_i^{in} are the set of constraint and set of cost function which are received from other nodes. As a result of observation and receiving, node i knows constraint $C_i = C_i^{out} \cup C_i^{in}$ and cost function $F_i = F_i^{out} \cup F_i^{in}$. If an observed constraint and a received constraint overlap, the constraints must be integrated in an appropriate manner.

Nbr_i is set of neighborhood nodes. It means set of names of the nodes in the strict sense. Nbr_i contains all nodes related by constraints in C_i. Degree of node i is defined to be $|Nbr_i|$. Deg_i^{in} is set of degrees which is received from neighborhood nodes. The information of degree is referred in the process of construction of tree.

Node i sends messages to related nodes. Each message includes logical time stamp on the sending. Clk_i^{in} is set of logical times which were most recently received from nodes in Nbr_i. Node i increments its logical time when its state is changed or new logical time is received.

$$clk_i \leftarrow \max(clk_i + \alpha, \max_j clk_j \in Clk_i^{in})$$

$$\alpha = \begin{cases} 1 & \text{state of node } i \text{ has changes} \\ 0 & \text{otherwise} \end{cases}$$

Node i must notify neighborhood nodes about the change of its state when the node increments its logical time. Messages of the notification is as follows.

(1) (**STS_1**,$c_{i,j}^{out}$,$f_{i,j}^{out}$,$|Nbr_i|$,clk_i)

(2) (**STS_2**,$|Nbr_i|$,clk_i)

(3) (**STS_3**,clk_i) (4) (**STS_4**,clk_i)

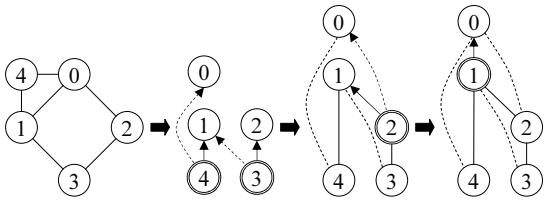

Figure 1. An example of construction of tree

Destination node of the above messages and information in the messages are as follows. (1) Destination node j is related by a constraint in C_i^{out}. The information is that new constraint was added or known constraint was changed. (2) Destination node j is related by a constraint not in C_i^{out} but C_i^{in}. The information is that the degree was changed. (3) Destination node j is a neighborhood node. The information is that the constraint which relates x_i to x_j was removed from C_i^{out}. If no constraint which relates x_j to x_i is contained in C_i^{in}, node j is removed from Nbr_i. (4) Destination node j is a neighborhood node. The information is that the state of node i has no changes except increase of clk_i.

When no changes of constraints are observed in all nodes, the notification converges. In that situation, clk_i, clk_j, and elements of Clk_i^{in} and Clk_j^{in} have equal values for all node j related to node i. If clk_i and elements of Clk_i^{in} have values equal to clk, node i is locally stable at clk. If node j is a neighborhood node of i, node j must receive message of clk from node i before it becomes stable at clk. From this condition, node i identifies its state Sts_i as $Sts_{i,clk}$ when it is locally stable at clk.

4 Construction of pseudo DFS tree

Nodes construct trees of constraint network for each problem. The tree is considered as a pseudo DFS tree of a connected component of constraint network. Basic idea of construction method for the tree is shown in [5]. We rearrange the method in order to apply it to dynamic problem. Following method also includes a rule for termination detection of the construction.

Let $Tree_{i,clk_i}$ denote description of tree for node i and its current logical time clk_i. Each node maintains only $Tree_{i,clk_i}$ of current logical time. Components of $Tree_{i,clk_i}$ are as follows.

- C_i, F_i, Nbr_i, Deg_i^{in}: copies of elements of Sts_{i,clk_i}.
- Nbr_upper_i, Nbr_lower_i: set of upper neighborhood nodes and set of lower neighborhood nodes.
- Deg_i: set of degrees of related nodes.
- Rel_i: set of counts of constraints for related nodes.
- $Children_i$: set of child nodes.
- $Ancestors_i$: set of ancestor nodes.
- $parent_i$: parent node.
- $num\_of\_leaf_i$: number of leaf nodes.

C_i, F_i, Nbr_i and Deg_i^{in} are copies of elements of Sts_{i,clk_i}. Note that Sts_{i,clk_i} may not be obtained for clk_i. On the other hand, node i maintains $Tree_{i,clk_i}$ of current logical time in any case. If Sts_{i,clk_i} is not obtained, the

copies are not available. Until Sts_{i,clk_j} is obtained, node i only receives **TREE** message as shown in the following. When current logical time of node i increases, $Tree_{i,clk_i}$ is replaced by $Tree_{i,clk'_i}$ for next logical time clk'_i.

Nbr_upper_i, $Nbr\_lower_i \subseteq Nbr_i$ are set of upper neighborhood nodes and set of lower neighborhood nodes. Each node in Nbr_i is classified using its name and degree in Deg_i^{in}. An appropriate ordered relation must be defined for name of nodes. Let deg_i, deg_j denote degree of i, j. If

$$deg_i > deg_j \vee (deg_i = deg_j \wedge i < j)$$

then i is prior to j. This max-degree heuristics improves parallelism of pseudo DFS tree. Deg_i is set of degrees of related nodes. Deg_i is union set $Deg_i^{in} \cup \left(\bigcup_{\forall j} Deg_j\right)$. Here Deg_j is the set of degree which is received from descendant node j. Rel_i is set of counts of constraint edges from sub-tree routed at i to its ancestor nodes. For node i, let $rel_{i,k} \in Rel_i$ denote the count of ancestor node k. If Nbr_upper_i contains node k, $rel_{i,k}$ is increased by one. For all child nodes, if Rel_j which is received from child node j contains $rel_{j,k}$, then $rel_{i,k}$ is increased by $rel_{j,k}$.

$Children_i$, $Ancestors_i$ and $parent_i$ are set of child nodes, set of ancestor nodes and parent node of node i. For all $j \in Ancestors_i$, j is prior to i. $parent_i$ is the lowest node which is contained in $Ancestors_i$. $num\_of\_leaf_i$ is number of leafs of sub-tree routed at i. If node i is a leaf node, then $num\_of\_leaf_i = 1$, otherwise $num\_of\_leaf_i$ is sum of $num\_of\_leaf_j$ which is received from child node j for all child nodes. The number of leaf nodes is referred in the process of termination of search.

When all child node of i is fixed, sub-tree routed at i is fixed. Node i sends **TREE** messages to $j \in Ancestors_i$, when its sub-tree is fixed.

(**TREE**,$parent_i$,$Ancestors_i$,Deg_i,Rel_i,
$num\_of\_leaf_i$,clk_i)

When node j receives **TREE** message from i, the message is processed as follows. $Ancestors_j$ and Deg_j are updated. If $parent_i = j$, then Rel_j and $num\_of\_leaf_j$ are updated, and i is added to $Children_j$. If clk_i of **TREE** message is less than clk_j of $Tree_{j,clk_j}$, the message is ignored.

When node i receives **TREE** message from its child node, the child node is added to $Children_i$. Each sub-tree routed at the child node includes at least one lower neighborhood node of i. If $Children_i$ contains all child nodes of i, $rel_{i,i} \in Rel_i$ is equal to number of the lower neighborhood nodes. From this condition, node i fixes its sub-tree if $rel_{i,i} = |Nbr\_lower_i|$. Sub-trees of leaf nodes are fixed immediately, and sub-trees are fixed up to root node.

The method does not generate a exact DFS tree, but a tree which has no constraint edges between sub-trees. If a parent node and its child node is not related by any constraint, a dummy constraint edge is inserted between the nodes. The constraint is not included in the evaluation of degree. An example of construction of tree is shown Figure 1. In the example, a dummy constraint edge is inserted between node 1 and node 2.

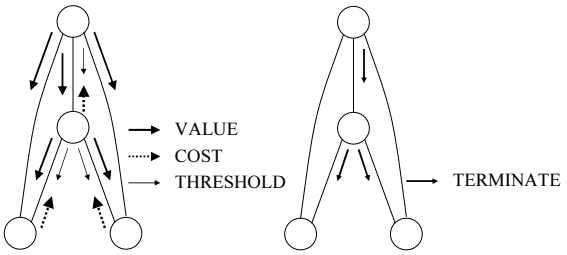

Figure 2. Adopt algorithm

$Tree_{i,clk}$ depends upon $Sts_{i,clk}$. If node i and node j are neighborhood, $Sts_{i,clk}$ and $Sts_{j,clk}$ have no contradiction between i and j. If $Sts_{i,clk}$ is never obtained, construction of tree for $Tree_{k,clk}$ is not completed. Even if node i fixed its sub-tree, node i discards $Tree_{i,clk_i}$ when its logical time is increased.

5 Execution of distributed search algorithm

A search algorithm based on Adopt algorithm[6] is applied to proposed framework. It is assumed that agents are placed in DFS tree before the algorithm is executed. Pseudo DFS tree shown in previous section is used instead of the DFS tree. See [6] for details of Adopt algorithm. In the following, brief explanation of the algorithm is shown.

Node i has $d_i \in D_i$, $threshold_i$, $currentContext_i$, $context_i(d,x)$, $lb_i(d,x)$, $ub_i(d,x)$ and $t_i(d,x)$. d_i is value of its variable. $threshold_i$ is cost allocated to sub-tree routed at i. Node i must satisfy the condition that actual cost of partial solution for its sub-tree is less than or equal to $threshold_i$. $currentContext_i$ is cache of partial solution of upper nodes. $context_i(d,x)$, $lb_i(d,x)$ and $ub_i(d,x)$ are caches of partial solution and corresponding upper/lower bound of cost. Here d is a value of its variable, and x is a child node. They are cached for all d and x. $t_i(d,x)$ is a cost which is allocated to child node x when $d_i = d$.

Node i evaluates local cost $\delta_i(d)$. $\delta_i(d)$ is defined to be sum of values of cost functions for all constraints between i and its upper neighborhood nodes. Node i evaluates UB_i and LB_i which are upper and lower bound of cost for sub-tree routed at i. They are calculated using $\delta_i(d)$, $lb_i(d,x)$ and $ub_i(d,x)$.

Processing of the algorithm is as follows. (1) Node i determines value of d_i according to LB_i and $threshold_i$. The value is sent to its lower neighborhood nodes (**VALUE** message). (2) Node i sends the value of UB_i, LB_i and $currentContext_i$ to its parent node (**COST** message). (3) Node i determines value of $t_i(d,x)$. The value of sent to its child node x when $d_i = d$ (**THRESHOLD** message). (4) Eventually, LB_i, $threshold_i$ and UB_i have equal values in root node. Then root node fixes d_i for optimal cost, and notify its child nodes of the termination (**TERMINATE** message). When the child nodes find optimal cost, they terminate in the same manner. Message paths of Adopt algorithm is shown Figure 2. Modification of the algorithm is shown in the following.

5.1 Identification of problems

The search algorithm is executed for current tree. When logical time of a node is increased, the node discards its tree and search, without any notification. As a result, execution of current search breaks down. On the other hand, nodes construct new tree and next search using the tree is started.

Adopt algorithm is a best-first search algorithm using backtracking. Its worst-case time complexity is exponential. In dynamic problem, available time for search is limited. In order to obtain quasi-optimal solution immediately, we modify the Adopt algorithm as shown in 5.4. When a quasi-optimal cost is obtained, root node terminates search. If the problem does not change until solution is fixed, the solution is refined in next search. A number of search processing are repeated for one problem of clk. Each search processing is identified using clk and sequence number slt. In first search for a problem, slt is initialized to zero. Otherwise value of slt is increased by one in every search processing.

Let $Adpt_{i,clk,slt}$ denote the description of Adopt algorithm for node i, logical time clk and sequence number slt. $Adpt_{i,clk,slt}$ includes variables of Adopt algorithm shown in the above. $Adpt_{i,clk,slt}$ also includes C_i, F_i, Nbr_i, Nbr_upper_i, Nbr_lower_i, $Children_i$, $parent_i$ and $num\_of\_leaf_i$ which are copies of elements of $Tree_{i,clk}$. Let $Adpt_i^{cur}$ denote the description of Adopt algorithm for current problem of node i. In order to identify corresponding problem, messages of Adopt algorithm are modified so that they include clk and slt. Node i processes only messages which corresponds to $Adpt_i^{cur}$. When node i receives message for previous problem, the message is ignored. Note that node i never receives any messages corresponding to problems which are more recent than $Adpt_i^{cur}$.

5.2 Modification of termination

As shown above, when current problem is changed, search processing for the problem disappears. However, termination of the algorithm must be synchronized in order to obtain complete solution. In our framework, the termination is synchronized using a method similar to two phase commitment algorithm. If sub-commitment of solution is completed in first phase, the solution is decided in second phase. If the root node detects change of its tree before first phase is completed, the solution is canceled in second phase. While nodes are waiting for the termination of its sub-tree, the nodes must prepare for next search. Moreover, in dynamic problem, previous solution should be backed up for solution reuse. Therefore three descriptions of Adopt algorithm are necessary for each node. Let $Adpt_i^{sub}$ and $Adpt_i^{cmt}$ denote descriptions for sub-commitment and backed up of previous solution.

Termination processing for $Adpt_i^{cur}$ is modified as follows. Modified **TERMINATE** message includes x_{root} and opt. x_{root} is the name of root node. opt is flag which indicates whether cost is optimal in root node.

[send (TERMINATE, $\cdots$, x_{root},*opt*,clk_i, slt_i)]
condition to send:
i is not a leaf node $\wedge$
termination condition of Adopt was satisfied.
destination: $x \in Children_i$
post-processing:
$Adpt_i^{sub} \leftarrow Adpt_i^{cur}$. $Adpt_i^{cur} \leftarrow Adpt_{i,clk,slt+1}$.

[when (TERMINATE, $\cdots$, x_{root},*opt*,clk_k, slt_k) is received]
Process rules of Adopt.
Record x_{root} and opt.
Record receiving of TERMINATE message.

[send (TERMINATE_OK,clk_i, slt_i)]
condition to send:
i is a leaf node $\wedge$
termination condition of Adopt was satisfied.
destination: x_{root}
post-processing: (same as **[send TERMINATE**])

In order to process decision or cancel of solution, following processing is added for $Adpt_i^{sub}$.

[when (TERMINATE_OK,clk_k, slt_k) is received]
Record number of **TERMINATE_OK** messages which were received.

[send (TERMINATE_COMMIT,clk_i, slt_i)]
condition to send:
(i is root node $\wedge$ number of **TERMINATE_OK** messages which were received = $num\_of\_leaf_i$)
$\vee$ (**TERMINATE_COMMIT** was received).
destination: $x \in Children_i$
post-processing:
$Adpt_i^{cmt} \leftarrow Adpt_i^{sub}$. Delete $Adpt_i^{sub}$.

[when (TERMINATE_COMMIT,clk_k, slt_k) is received]
Send **TERMINATE_COMMIT**.

[send (TERMINATE_CANCEL,clk_i, slt_i)]
condition to send:
(i is a root node $\wedge$ $Adpt_i^{cur}$ was updated)
$\vee$ (**TERMINATE_CANCEL** was received)
destination: $x \in Children_i$
post-processing: Delete $Adpt_i^{sub}$.

[when (TERMINATE_CANCEL,clk_k, slt_k) is received]
Send **TERMINATE_CANCEL**.

5.3 Condition to start search

$Adpt_{i,clk,0}$ depends upon $Tree_{i,clk}$. Therefore search for $Adpt_{i,clk,0}$ must be started after $Tree_{i,clk}$ is fixed. $Tree_{i,clk}$ is fixed in a bottom-up manner, and starting the search for $Adpt_{i,clk,0}$ in the same manner is possible. Each node must receive **VALUE** messages from its upper neighborhood nodes in order to compute the local cost. **COST** message is not sent until **VALUE** messages are received from all upper neighborhood nodes. When a node receives **VALUE** messages from all upper neighborhood nodes, search processing of its parent node is already started. Therefore the parent node is able to receive **COST** message.

When node i receives **TERMINATE** message for $Adpt_{i,clk,slt}$, the node sets $Adpt_i^{sub} = Adpt_{i,clk,slt}$ and $Adpt_i^{cur} = Adpt_{i,clk,slt+1}$. When a node decides its solution by receiving of **TERMINATE_COMMIT** message, its descendant nodes are already prepared for next search.

Therefore, if $slt > 0$, starting the search in a top-down manner is possible.

When node i has $Adpt_i^{sub}$, the node must wait for decision or cancel of its solution. On the other hand, the node receives messages for $Adpt_i^{cur}$. It is possible that the node receives messages without evaluation of its solution.

When opt flag shown in the above indicates optimal cost, search processing for $Adpt_i^{cur}$ is unnecessary.

Condition to start search is concluded as follows. If $Tree_{i,clk}$ for $Adpt_i^{cur}$ is already fixed and opt flag does not indicate the optimal cost, then search processing for $Adpt_i^{cur}$ is started. If node i has $Adpt_i^{sub}$, then node i does not send messages for $Adpt_i^{cur}$.

5.4 Modification of schedule of search

In order to obtain quasi-solution immediately, $threshold_i$ of Adopt algorithm should be initialized to a value which is greater than or equal to optimal cost. Therefore initial values of $threshold_i$ and $t_i(d, x)$ are set to infinity instead of zero. This modification restricts backtracking if cost of current solution is less than or equal to known upper bound. Only root node controls $threshold_{root}$ using an appropriate schedule. We use a simple schedule as follows. In the beginning of search processing, $threshold_{root}$ is set to an objective value. When $threshold_{root} = UB_{root}$ is satisfied, the search is terminated. Termination condition of Adopt algorithm is modified to exclude the case of $threshold_i = UB_i = \infty$.

5.5 Reuse of previous solution and cost

When $Adpt_i^{cmt}$ is available, initial value of d_i^{cur} of $Adpt_i^{cur}$ is set to value of d_i^{cmt} of $Adpt_i^{cmt}$. If $Adpt_{root}^{cmt} = Adpt_{root,clk,slt} \wedge Adpt_{root}^{cur} = Adpt_{root,clk,slt+1}$, that is, if the problem does not changes in root node, then initial value of $threshold_{root}^{cur}$ is determined using UB_{root}^{cmt} and LB_{root}^{cmt} of $Adpt_{root}^{cmt}$. We use $\max(LB_i^{cmt}, UB_i^{cmt} - \Delta)$ as initial value. Here Δ is a parameter.

When $Adpt_i^{cmt}$ is not available, initial value of d_i^{cur} and $threshold_i^{cur}$ are set to default values.

6 Evaluation

The behavior of the proposed method is evaluated using simulation. We evaluate cost of best solution which is obtained until the problem changes. Proposed method "dadpt" is compared with two distributed search methods which uses all nodes in the system. As complete method, we use modified Adopt algorithm "gadpt" which are identical with dadpt except that all nodes are placed in linear order. As approximate method, we use Distributed Greedy Repair method[3] "grpar". Note that complexity of computation of Distributed Greedy Repair method is less than proposed method. We assume that communication overhead is greater than computation.

In gadpt, if a parent node and its child node are not related by constraint edge, a dummy constraint edge is inserted between the nodes. Ordering of the nodes is determined in a random manner. In second phase of termination of search, root node multicasts messages to its descendants in order to reduce message cycle.

Table 1. Average cost of best solution(n=40, T=100)

d	alg.	P0		P1		P2		P3	
		mean #opt.	(var.)	m. #o.	(v.)	m. #o.	(v.)	m. #o.	(v.)
1	gadpt	13.5	(9.73)	13.1	(10.04)	13.2	(10.18)	13.4	(9.38)
		0		0		0		0	
	grpar	9.3	(6.34)	7.8	(4.68)	6.5	(4.65)	5.7	(4.20)
		0		0		0		1	
	dadpt	1.2	(1.25)	0.1	(0.22)	0.0	(0.03)	0.0	(0.00)
		36		90		97		100	
2	gadpt	26.8	(18.51)	26.7	(18.11)	27.0	(17.42)	27.0	(16.68)
		0		0		0		0	
	grpar	21.1	(15.64)	19.0	(13.07)	17.5	(11.83)	16.6	(10.11)
		0		0		0		0	
	dadpt	15.6	(10.83)	13.4	(13.63)	12.4	(12.47)	11.1	(10.57)
		0		0		0		0	
3	gadpt	40.2	(24.58)	40.3	(23.02)	40.4	(22.65)	40.4	(23.75)
	grpar	33.4	(15.47)	30.6	(15.65)	28.4	(15.03)	27.0	(13.51)
	dadpt	32.4	(40.74)	31.7	(47.45)	30.0	(45.54)	29.0	(50.96)

Table 2. Average cost of best solution(n=40, T=500)

d	alg.	P0		P1		P2		P3	
		mean #opt.	(var.)	m. #o.	(v.)	m. #o.	(v.)	m. #o.	(v.)
1	gadpt	5.6	(5.68)	3.7	(7.28)	2.8	(8.29)	2.3	(7.03)
		1		10		28		38	
	grpar	3.0	(2.49)	1.1	(0.85)	0.6	(0.58)	0.4	(0.41)
		2		29		53		72	
	dadpt	0.0	(0.00)	0.0	(0.00)	0.0	(0.00)	0.0	(0.00)
		100		100		100		100	
2	gadpt	17.8	(18.33)	16.8	(20.03)	16.5	(19.89)	16.3	(18.14)
		0		0		0		0	
	grpar	11.7	(7.44)	6.8	(4.31)	4.9	(3.25)	4.0	(2.30)
		0		0		0		1	
	dadpt	3.3	(4.67)	1.6	(2.74)	1.1	(1.71)	1.1	(1.65)
		11		36		62		63	
3	gadpt	32.0	(38.46)	31.2	(36.79)	30.3	(32.34)	30.1	(30.54)
	grpar	22.6	(10.99)	16.0	(8.97)	13.0	(5.17)	11.2	(4.59)
	dadpt	19.6	(18.47)	18.0	(18.24)	17.4	(16.41)	16.6	(15.25)

In dadpt and gadpt, we use $\Delta = (UB_i^{cmt} - LB_i^{cmt})/4$ as parameter for initialization of $threshold_i^{cur}$.

Distributed Greedy Repair method is a distributed version of centralized Greedy Repair method. The system consists of a "leader" node and other "member" nodes. Processing of the algorithm is as follows.

[Distributed Initialization] (1) Leader gives instruction to all members. (2) Each member notifies its neighborhood nodes of value of its variable. (3) Each member evaluates the cost of its partial solution, and notifies leader of the cost.

[Distributed Greedy Repair] (1) Leader gives instruction to a member in appropriate order. (2) The member executes hill climb method. Then the member notifies its neighborhood nodes of new value of its variable, and notifies leader of difference between new cost and old cost. (3) If total cost is optimal, leader gives instruction to all members so that they record current solution. (4) The above processing is repeated until termination condition is satisfied.

[Distributed Initialization] and [Distributed Greedy Repair] are repeated for the restart of search. The search is restarted if no improvement of total cost is obtained within

Table 3. Size of trees

n	d	#root nodes		max. #depth		max. #nodes	
		mean	(var.)	mean	(var.)	mean	(var.)
10	1	2.1	(0.30)	4.4	(0.40)	8.7	(0.64)
	2	1.0	(0.02)	6.8	(0.85)	10.0	(0.02)
	3	1.0	(0.00)	8.3	(0.56)	10.0	(0.00)
20	1	3.5	(1.08)	6.4	(0.79)	16.7	(2.56)
	2	1.1	(0.10)	12.1	(1.58)	19.9	(0.10)
	3	1.0	(0.00)	14.3	(1.58)	20.0	(0.00)
40	1	7.5	(3.78)	9.4	(1.81)	32.3	(4.90)
	2	1.5	(0.48)	19.0	(2.47)	39.5	(0.48)
	3	1.1	(0.20)	24.9	(4.12)	39.9	(0.20)

specified number of cycles. We use number of nodes as the number of cycles to restart. Termination condition for least lower bound is not used. It is assumed that multicast of messages is available. Instead, acknowledge messages for synchronization is assumed.

We use distributed graph coloring with 3 colors as in [6]. Each instance of dynamic problems is a sequence of 6 random binary DCOPs. Each DCOP is generated for n and d, where n is number of nodes and d is link density. Each DCOP has $n \cdot d$ constraint edges for n and d. Each constraint is assigned weight of 1. The problem is a maximum DCSP. Each DCOP is solved according to its ordering in the sequence. Cost of best solution for each problem is evaluated. Each problem is changed after T cycles. For each n, d and T, 10 sequences are generated. For each sequences, 10 trials are performed. The results are averaged for sequences and trials.

In dadpt, it is assumed that each constraint is observed by only one node. Number of message cycles is evaluated for total processing. In gadpt and grpar, it is assumed that observation and pre-processing of search is done using appropriate method. Number of message cycles is evaluated for search processing.

Average cost of best solution and number of optimal solution for each problem is shown in Table 1 and 2. In these tables, results for first four problems are shown (The problems are denoted by 'P' and number of ordering). In these results, improvement of solution rather overcomes change of problem. Therefore cost of solution decreases in most cases. $T = 100$ or 500 is not enough number of cycles for these search methods. However, in case of $d = 1$, all solution of proposed method reaches optimal solution until P3. Adopt algorithm is a complete algorithm. Therefore, for easy problems which has few number of nodes and link density, proposed method obtains optimal solution. On the other hand, even if the problem is not difficult, approximate method does not reach optimal solution in some cases. In case of $d = 1, 2$, each cost of proposed method is less than or almost equal to corresponding cost of approximate method. However, in case of $d = 3, n = 40$, each cost of proposed method is greater than corresponding cost of approximate method.

Size of trees which are generated by proposed method is shown in Table 3. Number of root nodes, depth of tree and maximum number of nodes are evaluated. If link density is small, depth of tree is small. In case of $d = 1$, some nodes have no constraint, and more than one root nodes are generated. However, owing to the method to generate problems, constraint network is not separated into parts of similar sizes. The separation of constraint network improves effectiveness of proposed method.

7 Conclusion

In this paper, we present a basic framework which applies DCOP algorithm using DFS tree to dynamic problem. Especially, if constraint network has more than one connected component, proposal method generates trees for each part of network. The trees perform separately, therefore it can be considered as a self organization of multi agent system.

For the sake of simplicity, stable solution and reuse of context of search which are studied in research of centralized dynamic CSP, were excluded from our discussion. Application of proposed method to practical problem will be included in our future work.

References

[1] J. Liu and K. Sycara, Exploiting problem structure for distributed constraint optimization, *Proc. 1st Int. Conf. on Multiagent Systems*, San Francisco, CA, 1995, 246–253.

[2] K. Hirayama and M. Yokoo, An Approach to Over-constrained Distributed Constraint Satisfaction Problems: Distributed Hierarchical Constraint Satisfaction, *Proc. 4th Int. Conf. on Multiagent Systems*, Boston, Massachusetts, 2000, 135–142.

[3] M. Lemaître and G. Verfaillie, An Incomplete Method for Solving Distributed Valued Constraint Satisfaction Problems, *Proc. AAAI97 Workshop on Constraints and Agents*, Providence, RI, 1997.

[4] M. Yokoo, E. H. Durfee, T. Ishida and K. Kuwabara, The Distributed Constraint Satisfaction Problem: Formalization and Algorithms, *IEEE Trans. Knowledge and Data Engineering 10*(5), 1998, 673–685.

[5] Y. Hamadi, Interleaved search in distributed constraint networks, *International Journal on Artificial Intelligence Tools 3*(4), 2002, 167–188.

[6] P. J. Modi, W. Shen, M. Tambe and M. Yokoo, Adopt: asynchronous distributed constraint optimization with quality guarantees, *Artificial Intelligence 161*(1–2), 2005, 149–180.

[7] G. Verfaillie and T. Schiex, Solution Reuse in Dynamic Constraint Satisfaction Problems, *Proc. 12th National Conference on Artificial Intelligence*, Seattle, WA, 1994, 307–312.

[8] R.J. Wallace and E.C. Freuder, Stable solutions for dynamic constraint satisfaction problems, *Proc. 4th International Conference on Principles and Practice of Constraint Programming*, Pisa, Italy, 1998, 447–461.

[9] B. Awerbuch, A New Distributed Depth-First-Search Algorithm, *Inf. Process. Lett. 20*(3), 1985, 147–150.